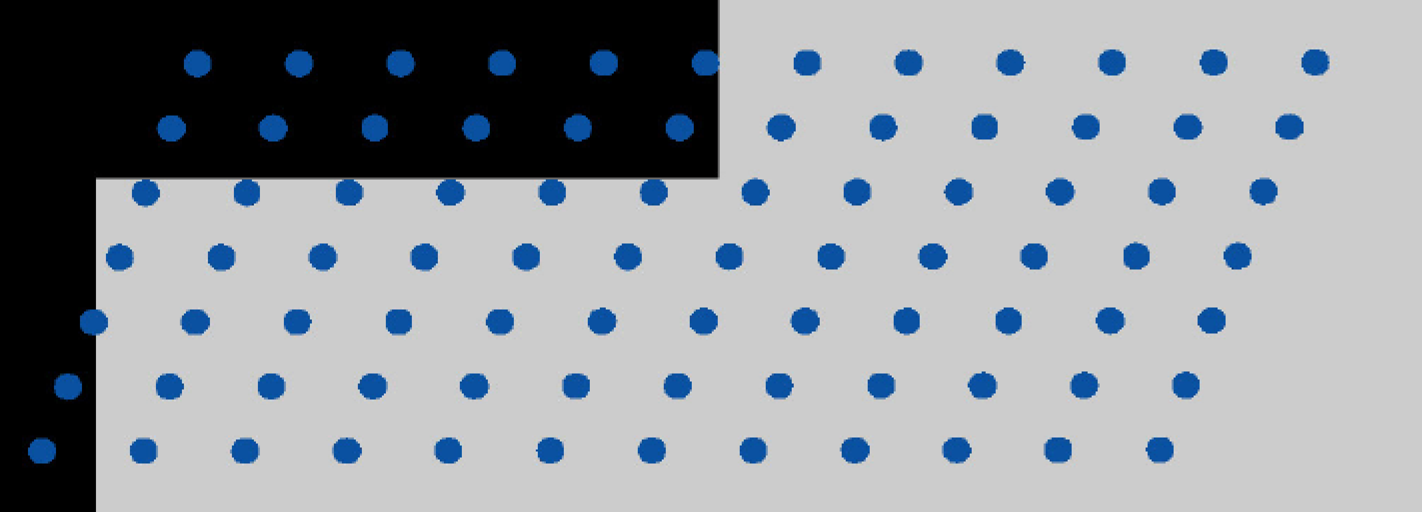
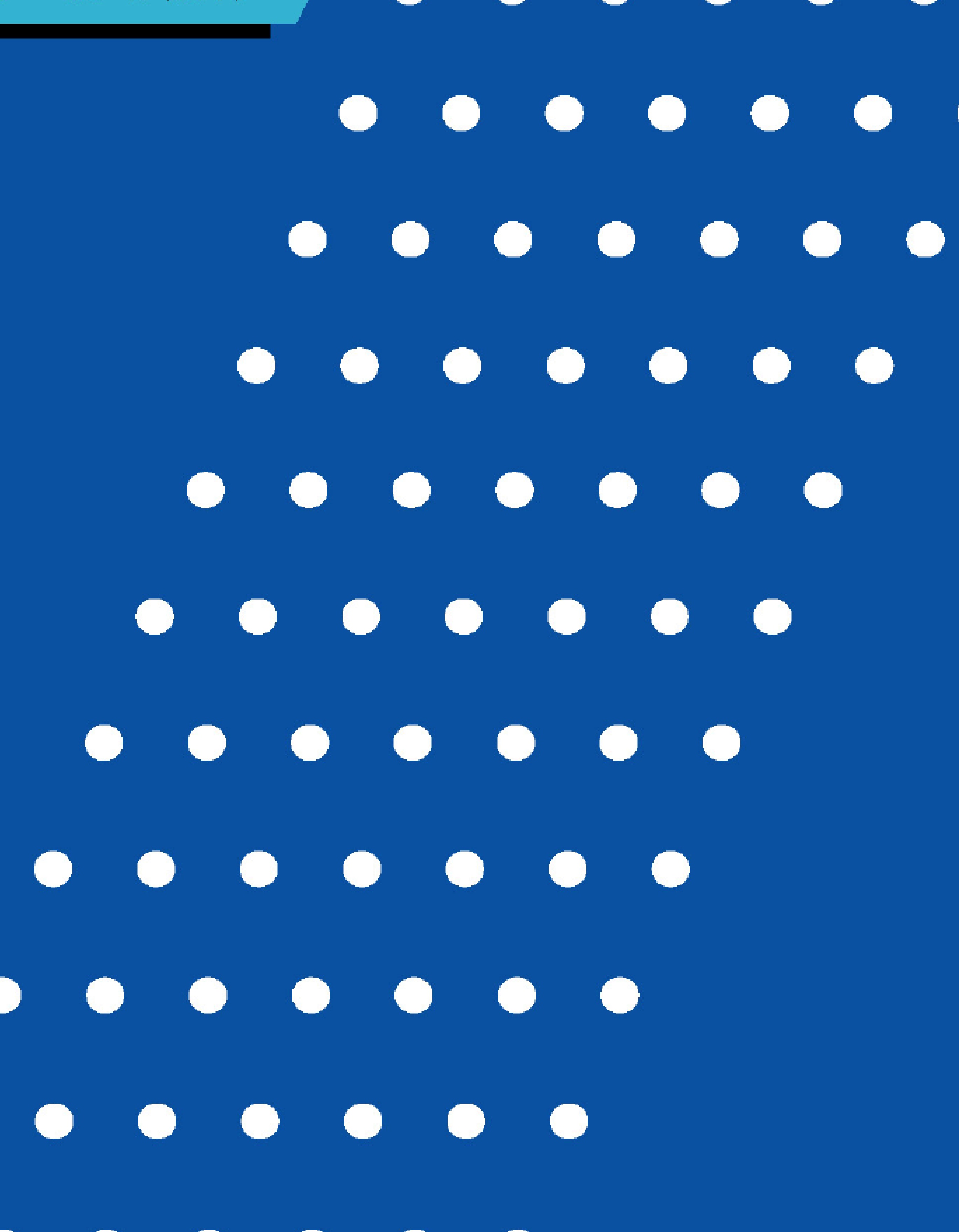


王英龙 张 伟 杨美红 主编

软件测试技术

教育部"软件工程课程体系研究"规划教材



清华大学出版社

软件工程系列教材

软件测试技术

王英龙　主　编
张　伟　杨美红　副主编

清华大学出版社

北京

内 容 简 介

本书首先从一个宏观的角度对软件测试做了准确定位，然后对软件测试做了总体论述并描绘了软件测试的一个全貌；随后，以软件测试生命周期这样一个基本的软件测试过程为线索，逐层深入地向读者解密软件测试的内容和技术，在本书的后面还论述了测试过程的组织与管理、测试的度量和过程改进及相关模型。理论与实践的紧密结合是本书的最大特点，本书精心准备了一个独立软件项目，自始至终保持将各类测试内容附着于该项目中，做到了测试的连续性和完整性，实景回放式的叙述方式使读者如临其境，带来有如真实的工作体验。

本书可以用作各类高等学校的软件测试专业及相关专业的教学用书，也可以作为认证考试的参考用书，供测试职业鉴定类的高级检验员、检验师，国家软件资格水平考试软件评测师、国际软件测试工程师使用。

图书在版编目（CIP）数据

软件测试技术/王英龙等主编．—北京：清华大学出版社，2009.9
（教育部“软件工程课程体系研究”规划教材）
ISBN 978-7-302-20878-5

Ⅰ．软…　Ⅱ．王…　Ⅲ．软件—测试—高等学校—教材　Ⅳ．TP311.5

中国版本图书馆 CIP 数据核字(2009)第 159254 号

责任编辑：张瑞庆　顾　冰
责任校对：焦丽丽
责任印制：李红英

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印　刷　者：北京密云胶印厂
装　订　者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：185×260　　**印　张**：26　　**字　数**：601 千字
版　　次：2009 年 9 月第 1 版　　**印　次**：2009 年 9 月第 1 次印刷
印　　数：1～3000
定　　价：37.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：033563-01

PREFACE

软件工程系列教材

前　言

在几年前的一次业内研讨会的间隙，我与几位老朋友在一起聊天，请他们谈一谈对软件测试的一些看法。这几位多是一些名企的资深项目经理或是技术骨干，他们对软件测试的评价却颇高，归结一下的话，可以用 12 个字概括：**不可或缺、卓有成效、大有可为**。伴随信息产业的发展，现代软件应用提出了更高需求，应用开发的规模和复杂度、系统集成的复杂度都超越以往，信息化风险陡增，客户对软件质量更加关注、要求更高。软件测试作为重要的质量保障手段，为软件项目的成功做出了自己的贡献，并为项目过程改进提供了真实的一线数据，甚至有人戏言患上了"测试依赖症"，这是其"不可或缺"的一面；随着行业对软件测试的持续关注和投入，软件测试发展迅速，测试手段和方法获得了长足进步，软件测试用事实说话，获得了业界广泛认可，软件测试"卓有成效"；但是另一方面，作为一个年轻的领域，有关软件测试的基础理论研究和实践仍然有待探索和积累，投身于这个领域，将获得更多的机遇和挑战，未来"大有可为"。

山东省计算中心多年来精耕于信息技术领域，承担了众多省内外科研项目，为软件测试的发展做出了重要贡献。同时，在对外提供软件测试服务过程中积累了丰富的测试经验，并借此成为业内重要的一极。"十年树木，百年树人"，软件测试的发展最终需要更多人才的培养，应业内同行的邀请，我们组织力量精心编撰了这本著作。在编写本书的过程中，我们力争详尽准确，对每一个知识点都极尽考究。但是限于时间和精力，书中一定还有不少瑕疵和纰漏，恳请读者朋友批评指正，我们将不胜感激。

为此，我们也希望能够有机会在未来继续对这本书持续完善，使它成为一本与众不同的、饱含生命力的"活书"，能够持续地从行业中汲取营养，并将知识远播。

全书共分5篇，系统地讲述了软件测试的一般过程和方法，从软件工程的高度对软件测试重新定位，从软件质量的视角对软件测试的价值重新诠释，注重基础理论讲解的同时也重视实践经验的总结，使全书既有知识面的宽度也有认识理解上的深度。本书编写过程中参阅研讨了大量测试著作以及一些培训和认证的资料，力争使本书知识覆盖面更加广泛，使之可以满足高等院校的日常教学要求，同时对于有进一步深造要求的企事业在职人员也大有裨益。

理论与实践的紧密结合是本书的最大特点，为了编写本书，我们精心准备了一个独立软件项目作为本书介绍方法的实践案例，使读者在本书层层推进、逐步深入的介绍过程中，完成了案例项目的整个测试，不知不觉地在学到理论知识的同时也获得了宝贵的实际项目体验。全书内容结构紧凑、前后呼应、一气呵成。

本书内容组织以行业内的国家标准为依据，是介绍相关标准最多、最全面的测试类书籍，使全书保持了一定的业界权威性，同时又不失于实践性。

另外，衷心感谢山东省信息系统测评工程技术研究中心团队成员张伟、王振操、韩明军、郭莹、张海燕、韩庆良等为本书最终成稿所做出的工作，在本书的编撰过程中他们付出很多，正是他们的辛勤劳动和汗水，最终凝聚成今天的这本著作。

最后，还应该感谢清华大学出版社的诸位朋友和同仁，感谢他们所提供的大力支持和协助，使得本书最终得以面世。

编　者
2009年7月

CONTENTS

软件工程系列教材

目录

第一篇　软件工程概述

第二篇 软件测试概述

第三篇　软件测试一般过程与方法

第五篇　软件测试工具及其应用

PART 1

第一篇

软件工程概述

本篇主要对软件工程的起源、软件工程相关基本概念以及软件过程能力评估模型CMM/CMMI进行简要介绍，目的是使读者对软件工程及其发展现状有一个概括性了解，从而有助于了解软件测试在软件工程中所处的地位及其发挥的重要作用。

本篇名词解释

软件(**software**)：是计算机系统中与硬件(hardware)相互依存的另一部分，它包括程序(program)、相关数据(data)及其说明文档(document)。

软件危机(**software crisis**)：落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象。

软件工程(**software engineering**)：软件工程是研究和应用如何以系统性的、规范化的、可定量的过程化方法去开发和维护软件，以及如何把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来。

软件开发过程模型(**software development process model**)：软件开发过程模型是软件开发全过程、软件开发活动以及它们之间关系的结构框架，它为软件项目的管理提供里程碑和进度表，为软件开发提供原则和方法。

过程(**process**)：“针对一个给定目的的一系列操作步骤”(IEEE-STD-610)，或为实现一个给定目标而进行的一系列运作步骤。过程具有这样一些的性质：时间性、并发性、

嵌套性和度量性等。

软件过程(software process):开发和维护软件以及相关产品涉及的一系列活动。其中软件相关产品包括项目计划、设计文档、源代码、测试用例和用户手册等。过程是活动的集合,活动是任务的集合,任务是把输入转换为输出的操作。

软件过程能力(software process capability):描述了在遵循一个软件过程后,所期待结果的界限范围。

软件过程成熟度(software process maturity):指一个特定的软件过程被明确地定义、管理、度量、控制,并且是有效的程度。成熟度可以用于指示企业加强其软件过程能力的潜力。当一个企业达到了一定的软件过程成熟级别后,它将通过制定策略、建立标准和确立机构结构使它的软件过程制度化,而制度化又促使企业通过建立基础设施和公司文化来支持相关的方法、实践和过程,从而使之可以持续并维持一个良性循环。

软件过程管理(software process management):对软件产品及其开发、维护和支持所涉及的工作过程进行管理。

软件过程改进(software process improvement):为了更有效的达到优化软件过程的目的而实施的改善或改变其软件过程的系列活动。

验证(verification):"验证"的目的在于按照需求(包括顾客需求、产品需求和产品构件需求)对产品和中间产品进行验证。"验证"过程强调验证准备、验证执行和确定纠正措施。"验证"是一种渐进的过程,因为它要在产品和工作产品整个开发过程中执行,即从对需求进行验证开始,然后是对推进中的工作产品进行验证,最后是对完成的产品进行验证。

确认(validation):"确认"的目的在于证明,产品或产品构件被置于其预定使用环境中时,适合于其预定用途。"验证"过程与"确认"过程看起来类似,但是它们处理的问题不同。"确认"是要证明所提供的(或将要提供的)产品适合其预计的用途,而"验证"则是要查明工作产品是否恰当地反映了规定的要求。换句话说,验证保证"做得正确",而确认则保证"做正确的东西"。

内部质量(internal quality):产品属性的总和,决定了产品在特定条件下使用时,满足明确和隐含要求的能力。

外部质量(external quality):产品在特定条件下使用时,满足明确或隐含要求的程度。

使用质量(quality in use):特定用户使用的产品满足其要求,以在特定的使用环境下达到有效性、生产率、安全性和满意度等特定目标的程度。

CHAPTER

第 1 章 软件工程的起源概述

软件工程诞生于 1968 年，是北大西洋公约组织（North Atlantic Treaty Organization，NATO）在讨论应对"软件危机"的对策的过程中提出的一个概念，意图通过工程化的生产方式使软件走上工业化道路，从而解决"软件危机"所带来的"危害"。本章从软件的发展阶段及其特点入手，解释了"软件危机"出现的原因，以及软件工程的出现及其发展。

1.1 软件的发展及特点

软件（software）是计算机系统中与硬件相互依存的另一部分，它包括程序、相关数据及其说明文档。用一个简单的公式来表示，即"软件＝程序＋文档＋数据"。

从一定意义上来讲，软件是伴随着世界上第一台计算机的诞生而同时出现的，并且随着计算机硬件的发展而不断寻求突破。而随着软件的复杂程度、规模、应用领域、开发方法等的不断完善，又进而对计算机的运算速度提出了更高的要求。因此，计算机软件与计算机硬件的发展是相互影响、相互制约的。

1.1.1 计算机硬件的发展

计算机硬件作为软件的载体，经历了以下几个发展阶段。

简单的计算工具及机械。人类出于农业生产和产品交换的目的，需要借助一些机械工具进行数字记录和简单的计算，如中国人在公元 13 世纪发明的算盘、苏格兰数学家在 17 世纪早期发明的一种用于计算乘法的"骨质拼条"，以及 17 世纪中期由法国科学家发明的齿轮式加减法器等。到 19 世纪 40 年代为止，所有的机械式计算机都没有硬件和软件之分，算法被固化在机械中，如果需要修改算法，只能重新制造计算机械。

第一台电子计算机。1946 年 2 月 15 日，世界上第一台通用电子数字计算机 ENIAC（埃尼阿克）宣告研制成功。ENIAC 共使用了 18 000 个电子管，另加 1500 个继电器以及其他器件，其总体积约 $90m^2$，重达 30T，占地 $170m^2$。ENIAC 的成功，是计算机发展史上的一座纪念碑，是人类在发展计算技术的历程中，到达的一个新的起点。这台耗电量为 140kW 的计算机，运算速度为每秒 5000 次加法，或者 400 次乘法，比机械式的继电器计算

机快 1000 倍。ENIAC 最初是为了进行弹道计算而设计的专用计算机。但后来通过改变插入控制板里接线方式来解决各种不同的问题，而成为一台可以进行多种科学计算和军事计算的通用计算机。ENIAC 程序采用外部插入式，每当进行软件中一项新的计算时，都要重新连接线路。有时几分钟或几十分钟的计算，要花几小时或 1～2 天的时间进行线路连接准备。这种外加式的计算机程序和极小的存储容量，使其尚未完全具备现代计算机的主要特征。

电子管计算机时代(1946—1959)。这一阶段的计算机主要用于科学计算，主存储器是决定计算机技术面貌的主要因素。当时，主存储器有水银延迟线存储器、阴极射线示波管静电存储器、磁鼓和磁心存储器等类型，通常按此对计算机进行分类。

晶体管计算机时代(1959—1964)。主存储器均采用磁心存储器，磁鼓和磁盘开始用作主要的辅助存储器。不仅科学计算用计算机继续发展，而且中、小型计算机，特别是廉价的小型数据处理用计算机开始大量生产。

1964 年，在集成电路计算机发展的同时，计算机也进入了产品系列化的发展时期。半导体存储器逐步取代了磁心存储器的主存储器地位，磁盘成了不可缺少的辅助存储器，并且开始普遍采用虚拟存储技术。随着各种半导体只读存储器和可改写的只读存储器的迅速发展，以及微程序技术的发展和应用，计算机系统中开始出现固件子系统。

20 世纪 70 年代以后，计算机用集成电路的集成度迅速从中小规模发展到大规模、超大规模的水平，微处理器和微型计算机应运而生，各类计算机的性能迅速提高。随着字长 4 位、8 位、16 位、32 位和 64 位的微型计算机相继问世和广泛应用，对小型计算机、通用计算机和专用计算机的需求量也相应增长了。

新一代计算机是把信息采集存储处理、通信和人工智能结合在一起的智能计算机系统。它不仅能进行一般信息处理，而且能面向知识处理，具有形式化推理、联想、学习和解释的能力，将能帮助人类开拓未知的领域和获得新的知识。

1.1.2 计算机软件的发展

正如计算机硬件在 60 年内发生了极大变化，同样对于计算机软件来说，从观念到目标等方面都发生了很大变化。

20 世纪 40—50 年代：这一时期，计算机主要用于科学或军事计算。计算机没有装入任何软件，只能识别二进制代码，计算程序被固化到计算机硬件中，完全由计算机硬件实现。软件开发可被称为程序编写。由于每个计算机系统的指令系统都是单独设计，没有规律，0 与 1 组成序列的含义难于辨认和记忆，因此编写、阅读、调试程序都非常困难。此时，软件尚未从计算机硬件中脱离出来。

20 世纪 60 年代：计算机应用不再局限于计算问题，应用范围扩大。计算机大规模进入商业、银行等领域，开始出现操作系统、汇编语言、高级编程语言等系统软件。这一时期编程的目的不再是关心计算机硬件的动作，而是要确定程序人员定义的动作序列。这种动作不是代表计算机的一条指令，而是一系列计算机指令的执行。计算机程序与计算机的指令系统和内部结构无关，它们更直接地接近人们要处理的问题，因此解决问题的规模与复杂性大为增加。

20 世纪 70 年代：这一时期，计算机应用主要是大量的各种类型的非数值计算为特征的商业事务应用，并设计到大量智能性很强的领域。这一时期也开启了计算机网络互联技术，操作系统、数据库系统都得到大力发展。

20 世纪 80 年代：这一时期最为突出的表现是大量的数据库应用系统。特别是关系数据库的发展，使各种商业事务及社会生活，如银行、保险、民航等各方面的问题都转化为文字、数字，以数据形式进入计算机及数据库，形成各种类型的计算机信息系统。计算机应用得到极大的发展。计算机系统不仅用于科学计算、增强军事及社会企业集团的功能，而且个人计算机的字处理程序等软件已广泛应用，提高了个人生产率。

20 世纪 90 年代，在经济全球化背景的推动下，计算机使用方式发生根本变化，网络化成为整个计算机业的长远发展趋势。计算机应用不仅要提高个人的生产率，而且是通过支持跨地区、跨部门、跨时间的群组共享信息协同工作来提高群组、集团的整体生产率。

综上所述，计算机软件的复杂程度从简单到复杂，计算机软件类型从以系统软件为主到以应用软件为主，计算机软件的用户从科学与军事机构到各类企事业单位，经历了极大的变化。现在，计算机软件已经是人们在生产、生活中必不可少的工具。

结合计算机硬件的发展历程可以看出，计算机硬件的发展已在一定程度上超越了计算机软件的发展，随着计算机网络、通信技术的飞速发展，在分布式计算平台上架设各类软件系统已经具备了良好的硬件基础。软件要实现更好的发展前景，已基本上不会受到硬件条件的限制。

1.1.3 计算机软件的特点

计算机软件是一种与传统的工业产品截然不同的产品。传统的工业产品通常在既定的生产线下进行物质原材料的组装而成，具有有形的外观，并且成本影响因素较稳定，具有一定的磨损和老化期。

软件同传统的工业产品相比，具有如下特性：

① 软件是一种逻辑产品，与物质产品有很大区别。软件产品是看不到摸不着的，因而具有无形性，是脑力劳动的结晶，是以程序和文档的形式出现的。软件保存在计算机存储器和光盘介质上，通过计算机的执行才能体现其功能和作用。

② 软件的主要生产过程是研制，其成本主要体现在软件的开发和研制上。软件一旦研发成功后，通过复制即可产生大量的软件产品。

③ 软件的生产线，没有原材料组装过程，其生产过程无法通过生产线、机器、流程等进行严格的工艺控制。

④ 软件在使用过程中，不存在磨损、老化的问题。但是为了适应硬件、外界环境、需求等的变化，需要对软件进行修改。这些修改不可避免的引入错误，导致软件失效率升高，从而使软件退化 。当修改的成本难以接受时，软件就被抛弃。

⑤ 软件的成本昂贵。软件开发需要投入大量、高强度的脑力劳动，成本非常高，风险也大。现在软件的开销已大大超过了硬件的开销。

⑥ 软件工作牵涉到很多社会因素。许多软件的开发和运行涉及机构、体制和管理方式等问题，还会涉及人们的观念和心理等因素。这些人的因素，常常成为软件开发的困难

所在，直接影响到项目的成败。

以上特点使得计算机软件的开发、管理与交付过程具有了极大的不确定性，因此也使得软件开发与管理的难度更加显著。

1.2 软件危机

软件规模越来越大，复杂度也越来越高，软件成本逐年上升，而客户的需求变化得愈加频繁，软件质量却没有可靠的保证，从而出现了一系列严重的问题。这些问题并不局限于那些被认为是"未能正常运行"的软件，在业已开发完成的软件中，依然存在着各类运行与维护的问题。软件危机在20世纪60年代末全面爆发。

落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象。

1.2.1 软件危机的表现

具体地说，在软件开发和维护的过程中，软件危机主要表现在以下几个方面。

1. 软件成本逐渐提高

在计算机发展初期，计算机系统主要应用于军事以及科学计算等非常狭窄的领域，研制经费主要由国家财政提供，研制者较少考虑研制代价问题。而随着计算机产业化的不断发展，计算机系统的应用领域不断扩大，伴之而来的是不断膨胀的软件系统研制费用支出。投资者开始将代价和成本作为重要的问题进行考虑。

软件作为计算机系统的重要组成部分，其在总投资中所占的比例也逐年增加。表1-1给出了美国空军计算机系统中软件所占总费用的比例。

表 1-1　美国空军计算机系统中软件所占总费用的比例

年份	软件在计算机系统总费用中所占比例	年份	软件在计算机系统总费用中所占比例
1955	18%	1980	80%
1970	60%	1985	85%
1975	72%		

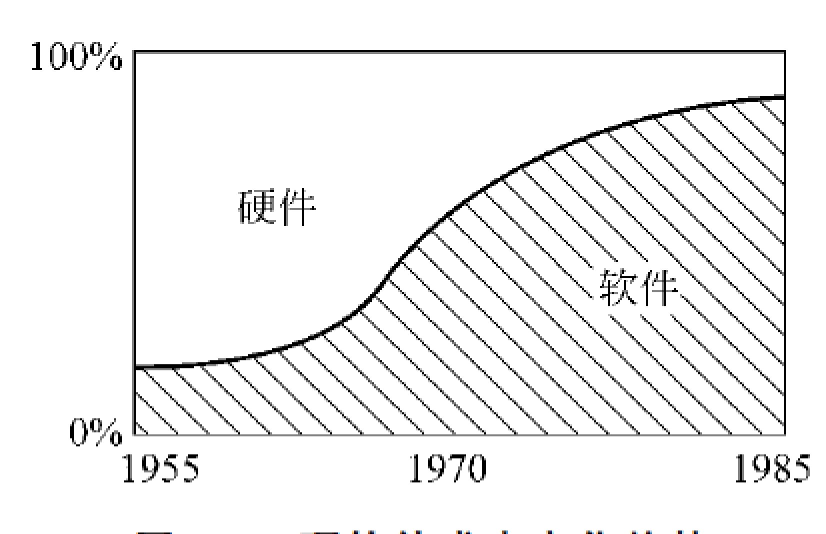


图 1-1　硬软件成本变化趋势

从表中所展示的趋势来分析，软件在整个计算机系统的总费用中所占的比例越来越大，而软件的应用领域也在不断地扩大，相应的对计算机硬件的要求也逐渐提高，而计算机硬件在整个计算机系统的总投资中所占的比例却急剧减少。这说明，软件价格的提高速度，已经大大超过了计算机硬件的价格的增长速度，成为计算机系统投资的重要成本对象(如图1-1所示)。

2. 软件质量难以保障

在生产制造领域，各类工业生产已经随着生产线的日趋完善而变得成熟，由这些生产线生产出的产品通常能够具备较好的质量，而此类推理在软件产业却无法实现。软件无法满足客户的需求，频繁发生更改，有时甚至造成严重的事故甚至灾难。如 1965 年至 1970 年，美国范登堡基地发射火箭多次失败，其中大部分原因是源于控制系统的应用程序缺陷所引起的系统故障；又如美国肯尼迪航空中心发射的一枚火箭，由于飞行计划程序中漏掉了一个连字符，致使火箭在飞离地面几十英里后开始翻转，从而遭到被迫炸毁的命运，1850 万美元付之东流。

以上列举的事件都是由于软件质量不合格造成的，究其原因有可能只是一处微小的错误引起的，但是造成的影响是深刻的，付出的代价是高昂的。从这些事件中，人们已经逐渐认识到软件产业有其自身特点和发展规律，对于影响软件质量的各类缺陷或错误，并非通过修改流水线，采用更好的原材料就能有效改进。软件是一项智力产品，其质量受到开发人员的技能、开发过程的规范程度以及社会和人的因素等多重影响。软件质量问题主要体现在以下几个方面：

- 软件需求得不到满足。
- 软件错误多，无法正常运行。
- 软件性能、可靠性、易用性差。

3. 开发进度难以控制

软件交付难成为开发方面临的重大问题。影响软件交付的重要因素是软件开发进度难以控制，这是因为软件是一种逻辑产品，为了完成一个复杂的软件，通常要建立庞大的逻辑体系，而同样的逻辑或同样的算法，可以用差别甚大的程序形式来实现。因此，软件开发过程是极难加以控制和规范的。

另一方面，软件开发过程中遇到的各种意想不到的情况层出不穷，项目组几乎每天都面临着软件过程中随时随地都会发生的变更，“计划不如变化快”成了软件从业者的一种无奈的表达，这种高频率的变更使软件开发过程很难保证按照预定的计划实现。

4. 运行维护成本增大

软件上线之后，运行维护任务便随之而来。软件维护人员往往到此时才发现，软件维护的工作量有可能会超过软件开发的工作量，而软件维护费用也随之逐渐加大。在软件的生命周期结束之前，软件的运维服务始终要伴随其左右。

软件维护任务之重在于，正式投入使用的商用软件，总是存在着一定数量的错误。随着时间的延伸，在不同的运行条件下，软件就会出现故障，就需要维护。这种维护与通常意义下的硬件维护是完全不同的。因为软件是逻辑元件，不是一种实物。对于软件错误或者软件缺陷的发现，需要从逻辑以及实现的角度去发掘。

一般一个软件的维护性工作包括改正性、适应性、增强性等三种类型。

- 改正性维护：主要是改正软件缺陷，纠正功能、性能等方面的错误。
- 适应性维护：主要用于适应数据环境、硬件及操作系统及系统移植工作。
- 增强性维护：主要用于提高处理效率、提高性能，满足新的需求，或使之使用方

便，增加及改善输出信息等。

在以上三种维护性工作中，占总的维护工作量比例最大的是增强性维护，约占 50% 左右。原因在于，软件用户通常认为软件易于修改、容易扩充，改变程序中的几条语句，程序功能就可以发生很大的变化，因此经常会提出要求进行软件的改造或升级。而随着软件系统逐渐变得庞大的同时，经常会出现"纠正一个错误而引发另外一连串的错误"的情况，因而软件维护工作量也随之不断增长。

软件维护工作量的增长使其在整个软件生命周期中所占的比例逐渐增大，软件开发机构不得不在软件交付使用之后，还要配备一定的人力、物力去维护系统的运行，从而导致软件运行维护的成本急剧提高。

5. 软件生产率的提高无法超越硬件与需求的增长

软件成本在计算机系统总成本中所占的比例居高不下，且逐年上升。由于微电子学技术的进步和硬件生产自动化程度不断提高，硬件成本逐年下降，性能和产量迅速提高。然而软件开发需要大量人力，软件成本随着软件规模和数量的剧增而持续上升。从美、日两国的统计数字表明，1985 年度软件成本大约占总成本的 90%。软件开发生产率提高的速度远远跟不上计算机应用迅速普及深入的需要，软件产品供不应求的状况使得人类不能充分利用现代计算机硬件所能提供的巨大潜力。

总之，可以将软件危机归结为成本、质量、生产率等几个方面的问题。对于每个软件开发组织来说，为了开发新的软件要组织大批软件人员投入一个很难控制、很难有所预见的过程中。许多项目开发的结果往往由于软件质量问题造成巨大损失。

1.2.2 软件危机的形成原因

1. 软件规模越来越大

随着计算机应用领域的扩大和深入，软件的规模也越来越呈现急速增长的趋势。从小型的桌面软件到大型的网络分布式软件，从几百行的程序到几千万行的复杂应用，软件规模已经在近 30 年中扩大了 100 倍。软件规模的增大，意味着软件开发过程的加长和复杂度的提高，意味着软件维护工作量和复杂度的提高，意味着算法复杂度的提高，意味着更加难以控制的软件质量。

以美国宇航局的软件系统为例：

- 1963 年：水星计划系统 200 万条指令。
- 1967 年：双子星座计划系统 400 万条指令。
- 1973 年：阿波罗计划系统 1000 万条指令。
- 1979 年：哥伦比亚航天飞机系统 4000 万条指令。

假设一个人一年生产一万条有效指令，那么是否 4000 人生产一年，或 400 人生产 10 年就能完成任务呢？答案是否定的。一万条指令的复杂度绝不仅仅是 100 条指令复杂度的 100 倍。

2. 软件复杂度越来越高

软件不仅在规模上快速的发展扩大，而且其复杂性也急剧地增加。软件复杂度主要

体现在软件接口复杂度上，一个软件系统中的各个子系统之间以及系统与外部系统之间的接口的数量和复杂程度在一定程度上反映了软件系统的复杂程度。研究表明，对于有 N 个子系统的软件系统，就可能有 $N(N-1)/2$ 种关系，而这些关系都需要有接口来实现。考虑到几千万行系统中众多的子系统或模块，则问题将变得十分严重。可以说，大型系统或军用软件费用超支的主要原因已被认为根源于系统的复杂性。

从计算机的应用领域的角度来讲，计算机应用从数值计算开始，发展到现在的大型业务系统(如金融、航空等领域)，均会涉及到几千万终端用户的交互操作，其复杂性可想而知。

3. 软件需求变化频繁

软件需求的变化是软件项目基本无法回避的一个现实。软件需求的不确定性为软件项目的成功造成了不小的隐患。许多软件项目由于频繁的需求变更而不得不投入大量的人力资源进行维护，以至于无法收回成本。

4. 软件技术进步落后于软件需求的增长

几十年来，从软件开发的方法、技术、工具等方面都出现了较大的跨越式进步，但这种进步的增长速度远远低于了软件需求的增长和复杂度的提高(如图 1-2 所示)。许多新的需求被提出来，而是用现有的技术却无法解决，因此需要花费成本去探索和学习新的技术，这种学习有时会需要几年甚至十几年的时间，无形中为软件行业的发展造成了“瓶颈”。

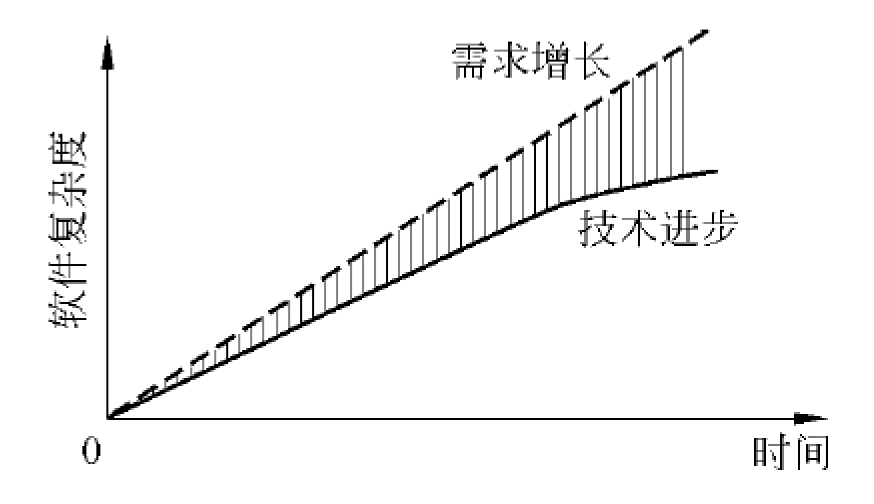


图 1-2 软件技术进步落后于客户需求

5. 软件自身的特点

软件自身的特点也是引发软件危机的一个重要原因。软件不同于硬件，它是计算机系统中的逻辑部件而不是物理部件；软件样品即是产品，试制过程也就是生产过程；软件不会因使用时间过长而“老化”或“用坏”；软件具有可运行的行为特性，在写出程序代码并在计算机上试运行之前，软件开发过程的进展情况较难衡量，软件质量也较难评价，因此管理和控制软件开发过程十分困难；软件质量不是根据大量制造的相同实体的质量来度量，而是与每一个组成部分的不同实体的质量紧密相关，因此，在运行时所出现的软件错误几乎都是在开发时期就存在而一直未被发现的，改正这类错误通常意味着改正或修改原来的设计，这就在客观上使得软件维护远比硬件维护困难；软件是一种信息产品，具有可延展性，属于柔性生产，与通用性强的硬件相比，软件更具有多样化的特点，更加接近人们的应用问题。随着计算机应用领域的扩大，99%的软件应用需求已不再是定义良好的数值计算问题，而是难以精确描述且富于变化的非数值型应用问题。因此，当人们的应用需求变化发展的时候，往往要求通过改变软件来使计算机系统满足新的需求，维护用户业务的延续性。

6. 其他因素

软件开发离不开人的开发行为，因此人的因素也是形成软件危机的一个影响方面，主要表现在：

（1）软件产品是人的思维结果，因此软件生产水平最终在相当程度上取决于软件人员的教育、训练和经验的积累。

（2）对于大型软件往往需要许多人合作开发，甚至要求软件开发人员深入应用领域的问题研究，这样就需要在用户与软件人员之间以及软件开发人员之间相互通信，在此过程中难免发生理解的差异，从而导致后续错误的设计或实现，而要消除这些误解和错误往往需要付出巨大的代价。

（3）由于计算机技术和应用发展迅速，知识更新周期加快，软件开发人员经常处在变化之中，不仅需要适应硬件更新的变化，而且还要涉及日益扩大的应用领域问题研究；软件开发人员所进行的每一项软件开发几乎都必须调整自身的知识结构以适应新的问题求解的需要，而这种调整是人所固有的学习行为，难以用工具来代替。

软件生产的这种知识密集和人力密集的特点是造成软件危机的根源所在。

1.2.3 软件工程的提出

1968年10月，北大西洋公约组织的科学委员会在德国加尔密斯开会讨论软件可靠性及软件危机的问题，产生了关于“软件工程”的加尔密斯报告，首次提出了“软件工程”这一概念。

分析带来软件危机的原因，宏观方面是由于软件日益深入社会生活的各个层面，对软件需求的增长速度大大超过了技术进步所能带来的软件生产率的提高。而就每一项具体的工程任务来看，许多困难来源于软件工程所面临的任务和其他工程之间的差异，以及软件和其他工业产品的不同。

在软件开发和维护过程中，之所以存在这些严重的问题，一方面与软件本身的特点有关，例如，软件开发质量难以评价，管理和控制软件开发过程相当困难，软件维护意味着改正或修改原来的设计；另外，软件的显著特点是规模庞大、复杂程度高，在开发大型软件时，要保证高质量，十分复杂困难。另一方面与软件开发和维护方法不正确有关，这是主要原因。

为了消除软件危机，通过认真研究解决软件危机的方法，认识到软件工程是使计算机软件走向工程化的科学途径，逐步形成了软件工程的概念，开辟了工程学的新兴领域软件工程学。软件工程就是试图用工程、科学和数学的原理与方法研制、维护计算机软件的有关技术及管理方法。在国标GB/T 11457—2006《信息技术 软件工程术语》中，有关软件工程的描述是这样的，软件工程是应用计算机科学理论和技术以及工程管理原则和方法，按预算和进度，实现满足用户要求的软件产品的定义、开发、发布和维护的工程或进行研究的学科。

软件工程包括3个要素，即方法、工具和过程。方法是完成软件工程项目的技术手段；工具支持软件的开发、管理、文档生成；过程支持软件开发的各个环节的控制、管理。

软件工程的进步是近几十年软件产业迅速发展的重要原动力。从根本上说，其目的是研究软件的开发技术，软件工程的名称意味着用工业化的开发方法来替代小作坊式的开发模式。但是，几十年的软件开发和软件的发展的实践证明，软件开发即不同于其他工业工程，也不同于科学研究。软件不是自然界的有形物体，它作为人类智慧的产物有其本

身的特点,所以软件工程的方法、概念、目标等都在发展,有的与最初的想法有了一定的差距。但是认识和学习过去和现在的发展演变,真正掌握软件开发技术的成就,并为进一步发展软件开发技术,以适应时代对软件的更高期望是有极大意义的。

软件工程的核心思想是把软件产品(就像其他工业产品一样)看做是一个工程产品来处理。把需求计划、可行性研究、工程审核、质量监督等工程化的概念引入到软件生产当中,以期达到工程项目的三个基本要素:进度、经费和质量的目标。同时,软件工程也注重研究不同于其他工业产品的一些独特特性,并针对软件的特点提出了许多有别于一般工业工程技术的一些技术方法。代表性的有结构化的方法、面向对象方法和软件开发模型及软件开发过程等。

从经济学的意义上来说,考虑到软件庞大的维护费用远超软件开发费用,因而开发软件不能只考虑开发期间的费用,而应考虑软件生命周期内的全部费用。因此,软件生命周期的概念就变得特别重要。在考虑软件费用时,不仅仅要降低开发成本,更要降低整个软件生命周期的总体成本。

本章小结

软件危机是计算机硬件与软件发展到一定阶段的必然产物,主要表现在软件成本逐年上升、软件质量难以保障、开发进度难以控制等几个方面。究其产生的原因,大致包括软件复杂性、软件规模、软件需求变化、软件自身特点以及人的因素等几个方面。为了应对软件危机,研究者们提出了“软件工程”的概念,希望借工程化的方法来进行软件产品的生产,以提高生产率和软件质量。

CHAPTER

第2章 软件工程概览

软件工程是软件行业的工程学科，主要研究如何使用工程学的方法来“建造”软件。软件工程包括方法、工具和过程三要素，共同构筑了软件质量的基石。通过几十年的探索和实践，业界已经形成了多种软件开发过程模型，并形成了多项相关标准，为软件工程提供了有效的指导。

2.1 软件工程的概念

工程，原指土木建筑或其他生产、制造部门用较大而复杂的设备来进行的工作，现在泛指通过系统的生产、制造、策划行为，形成一定的工作成果的过程。实施一项工程的产出结果可能是一幢建筑物、一台机器设备或者是一项政策措施。工程具有目标性、协作性、周期性等特点，是“由一群人为达到某种目的，在一个较长时间周期内进行协作活动的过程”。

软件工程是一类求解软件的工程。它应用计算机科学、数学及管理科学等原理，借鉴传统工程的原则、方法，创建软件以达到提高质量、降低成本的目的。其中，计算机科学、数学用于构造模型与算法，工程科学用于制定规范、设计范型、评估成本及确定权衡，管理科学用于计划、资源、质量、成本等管理。

软件工程是一门指导计算机软件开发和维护的工程学科，同时也是一门交叉学科，很多学者、组织机构都分别给出了自己的定义。

① Boehm。运用现代科学技术知识来设计并构造计算机程序及为开发、运行和维护这些程序所必需的相关文件资料。

② IEEE。软件工程是：

- 将系统化的、严格约束的、可量化的方法应用于软件的开发、运行和维护，即将工程化应用于软件；
- 在前一项所述方法的研究。

③ Fritz Bauer。建立并使用完善的工程化原则，以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法。

目前比较认可的一种定义认为：软件工程是研究和应用如何以系统性的、规范化的、

可定量的过程化方法去开发和维护软件，以及如何把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来。

软件工程学出现的直接诱因是软件危机。但事实上，计算机应用软件的规模和复杂性不断增加，软件作坊式的开发技术已不能与之相适应。因此，软件工程学的出现是软件开发技术发展的必然结果，软件开发的大生产规模特性注定了要用工程的方式进行，即对项目开发人员进行严密的组织管理，良好的协同配合。

软件工程的基本目标主要包括质量、成本、进度等三大方面，即付出较低的开发成本，在规定的时间内及时交付达到预期的软件功能、具有较好的软件性能、易于移植的软件产品，并需要较低的维护费用。

2.2 软件工程的要素

软件工程包括三个要素：方法、过程和工具。

软件工程方法为软件开发提供了“如何做”的技术，是指完成软件开发的各项任务所采取的技术路线、策略和措施。它包括了多方面的任务，如项目计划与估算、软件系统需求分析、数据结构、系统总体结构的设计、算法过程的设计、编码、测试以及维护等。

软件工具为软件工程方法提供了自动的或半自动的软件支撑环境。目前，已经推出了许多软件工具，这些软件工具集成起来，建立起称之为计算机辅助软件工程(CASE)的软件开发支撑系统。CASE 将各种软件工具、开发机器和一个存放开发过程信息的工程数据库组合起来形成一个软件工程环境。

软件工程的过程是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和协调变化所需要的管理以及软件开发各个阶段完成的里程碑。

软件工程是一种层次化的技术。任何工程方法(包括软件工程)必须以有组织的质量保证为基础。全面的质量管理和类似的理念刺激了不断的过程改进，正是这种改进使更加成熟的软件工程方法的不断出现。支持软件工程的根基就在于对质量的关注。

2.2.1 软件工程方法

经过 30 多年的研究与实践，人们已经成功地摸索和建立了多种软件工程方法，如结构化方法、面向对象方法、形式化方法、基于构件的方法、基于 Agent 的方法、基于敏捷技术的方法等。这些方法在自身的发展过程中又不断吸收其他方法和技术的长处，导致新技术新方法层出不穷，成为现代软件工程发展过程的亮点，从而不断丰富和发展了软件工程的理论与实践。

1. 结构化方法

结构化方法又被称为模块化方法，是最早的也是最具里程碑意义的一种软件工程方法，适用于规模较小、复杂度不高的模块化软件系统的设计与开发。结构化方法的特点是将系统分解为具有层次结构的模块或过程，在设计和实现各个模块的时候不需要考虑其

他模块的内部实现细节，而只需考虑本模块的实现和与其他模块的接口。从而采用自顶向下、逐层分解、逐步求精的方法，将待解决问题逐层分解，直到每个小问题都足够简单并易于处理。最后形成整个问题或系统的解空间。

结构化方法经过 20 多年的发展，已经形成了一套比较成熟的理论，它对软件工程的重要意义在于：第一，使人们认识到系统分析和设计比程序编写更加重要；第二，模块化思想对系统的分解合成、抽象与信息隐藏，易于使系统开发的脉络清晰明确；第三，数据流图、结构图和流程图等工具使用方便、操作性强。结构化方法也存在一些弊端，如模块的重用性差、缺乏信息隐蔽机制等。结构化方法为以后各类软件工程方法的发展奠定了一定基础。

2. 面向对象方法

面向对象的软件开发方法在 20 世纪 60 年代末期开始提出，经过近 20 年发展，这种技术才逐渐为人们所认识。到了 20 世纪 90 年代前期，面向对象的软件工程方法学已经成为人们开发大型软件项目时首选的模型。

面向对象方法学认为客观世界中的所有事物都可以被抽象为对象，而各个对象之间又存在着各种关系。具体到软件系统，面向对象方法学认为软件系统也是由对象组成的，对象将实体的属性和操作封装在一起，将更加符合人类认识事物的规律。

面向对象方法的核心概念是“类”。它是对具有相同数据和相同操作的一组相似对象的定义，也就是说，类是对具有相同属性和行为的一个或多个对象的抽象描述。面向对象技术中的关键机制是关于类的封装性、多态性和继承性。封装是把相关数据和过程封装在一起，使其成为一个相对独立的实体；多态指对象对同一消息可以有不同响应的能力；继承表示一种在父类和子类之间共享信息同时又可以使其具有个性的机制。

采用面向对象技术开发的软件工程项目具有稳定性好、可重用性高、可维护性强等优点，因此该技术比较适合于大型软件系统的开发。20 世纪 80 年代末以来，出现了诸如 Coad/Yourdon、Booch、OMT、Jacobson、Wirfs-Brock 等几十种面向对象方法，对面向对象软件系统的分析、建模、部署等方面都提供了一些重要的有建设性的方法。尤其值得一提的是统一建模语言（Unified Modeling Language，UML）的出现，它从 Booch、OMT 和 Jacobson 等方法和工程实践中吸收了许多经过实际检验的概念和技术，统一了符号体系，给软件工程带来了新气象，是软件工程发展过程中一件具有里程碑性质的新进展。

3. 形式化方法

计算机软件系统的构造具有很大的不确定性，从系统规划、需求到系统的实现，均有可能由于不同的方法、工具、算法、人员而发生变化。因此如果能够使用如数学等此类严谨的方法对软件进行定义，那么能够在一定程度上控制这种不确定性。

用于开发计算机系统的形式化方法是描述系统性质的基于数学的技术，这样的形式化方法提供了一个框架，可以在框架中以系统的而不是特别的方式刻画、开发和验证系统。形式化方法的本质是基于数学的方法来描述目标软件系统属性的一种技术。

根据形式化的程度，可以把软件工程划分为非形式化、半形式化和形式化三类。使用自然语言描述需求规格说明，是典型的非形式化方法。使用数据流图或实体关系图等图

形符号建立模型,是典型的半形式化方法。业界广为流传的典型的形式化方法有 Z 方法、Larch 方法、B 方法等。

形式化方法作为软件工程的基础性研究,在过去 20 多年里一直备受关注。近十几年来,国外对形式化方法在软件开发中的研究与应用开展了大量的实践工作,形式化方法已不再仅限于纯学术性研究,而是已经开始被工业界接受并用于实际开发的系统。国外已经出现不少包括形式化方法、形式化语言和形式化工具在内的比较成熟的形式化系统。

4. 基于构件的方法

软件产业化趋势导致了软件构件的产生。能够像硬件系统那样,将部分软件组合起来构建软件系统,一直是软件行业多年来追求的目标。尤其对于多数行业软件(如金融、税务、电子政务等),若能充分利用已有的软件构件,将会大大提高生产效率,减少大量的重复劳动。构件(component)是可复用的软件组成成分,是可以独立地制造、分发、销售和装配的二进制软件单元,是可执行软件的一个物理封装,它有定义良好的接口,可被用来构造其他软件。从低端的角度讲,构件可以是被封装的类、一些功能模块,从高端的角度讲,构件也可以是软件框架、软件构架(或体系结构)、文档、设计模式等。从广义上讲,软构件技术是基于面向对象的,以嵌入后马上可以使用的即插即用型软件概念为中心,通过构件的组合来建立应用的技术体系。狭义上讲,它是通过构件组合支持应用的开发环境和系统的总称。

基于构件的软件工程(Component-Based Software Engineering, CBSE)出现,给软件生产工厂化和自动化提供了可能。应用构件技术来开发软件项目是 20 世纪 90 年代开始出现在软件工程领域的热点。

基于构件的软件工程过程涉及三个主要子过程:构件开发、构件管理、基于构件的应用组装。构件开发包括构件建模、制作、获取和测试等过程。构件管理涉及构件库数据模型、构件的分类检索策略及构件 broker 等工作。基于构件的应用组装所要研究的内容包括:软件体系结构、源代码级的组装技术、运行级的组装技术、支持即插即用的技术、面向 CORBA、OLE、JavaBean 的辅助开发工具、应用系统演化及构件的灵活替换和升级。

构件技术对于软件工程的发展有着重要意义,它从根本上改变了软件生产方式,提高了软件生产的效率和质量,提高了软件复用程度。开发者可以摆脱编程的细节问题,将更多的注意力放到业务流程和业务逻辑上去。同时,用构件技术开发的系统灵活,便于维护和升级,进一步降低了对系统开发者的要求。目前,CBSE 的应用日益广泛,发展前景看好。

5. 基于 Agent 的方法

Agent 是近年来计算机科学领域中出现的一个新概念。中文译名有“智能体”、“智体”、“代理”、“主体”等。面向 Agent 的观点认为现实世界是由许多自主的或非自主的实体组成的,它们按照各种关系组织起来,彼此间进行各种交互(协作、协调、协商等)与通信,完成各种复杂的任务。它们可由多个 Agent 来模拟,合作、交互实现一些高级的功能. 相对于面向对象方法来说,面向 Agent 的观点对现实世界的模拟更自然、更直观、更容易为一般人所理解,因此采用面向 Agent 技术时,对软件的需求分析简单,更容易理

解，系统设计更方便，更简单。

从软件工程发展过程来看，面向 Agent 软件工程，无疑是当代软件工程思想（面向对象、构件等）的一种进步。人们由 OOA（Object Oriented Analysis）方法提出 AOA（Agent-Oriented Analysis），由 OOD（Object Oriented Design）提出了 AOD（Agent-Oriented Design），由 OOP（Object Oriented Programming）提出 AOP（Agent-Oriented Programming）。随着网络技术的发展，尤其 Internet 的应用及其固有的开放、复杂、异质、动态、分布等特性，为 Agent 技术的应用和发展提供了更好的平台。

6. 基于敏捷技术的方法

敏捷软件开发，特别是敏捷建模，对大多数的软件组织来说都是新生事物，是一种新的工作方式。敏捷方法汲取了众多轻型方法的“精华”，恰当地表达了这些轻型方法的最根本之处。首先，敏捷方法强调适应，而非预测重型方法花费大量的人力物力，试图对一个软件开发项目在很长的时间跨度内作出详细周密的计划来指导长期的工作，而一旦情况变化，计划就不再适用。因此重型方法本质上是抵制变化的，而敏捷方法则强调适应变化，它能够适应变化的过程，甚至能允许改变自身来适应变化。其次，敏捷方法以人为中心，是“面向人”的（people-oriented）而非“面向过程”的（process-oriented）。实施敏捷过程的一个关键之处是让开发人员选择并接受一个适应性过程而非强加给他一个过程。这一点在极端编程（Extreme Programming，XP）中特别明显，因为这需要很强的自律性来运行这个过程。

同其他方法一样，敏捷方法也需要建模。不同的是，敏捷建模（Agile Modeling，AM）是一组软件建模阶段的指导性原则，是针对基于软件系统的有效建模和编写文档的一个混乱而有序的、基于实践的方法，是 Scott W. Ambler 在众多敏捷方法学的基础上发展起来的实践的抽象和总结。

综上所述，软件工程技术和方法在不断发展。就软件开发方法而言，早已从面向数据流、面向数据结构和面向过程的设计方法过渡到面向对象的设计方法。为了设计出大规模、复杂度高的软件，必须有更高水准的结构设计技术。因此，20 世纪 90 年代以来又提出了一种能够对软件的所有结构进行设计的“软件机能构造”技术（即面向角色程序设计技术）。为了提高“软件机能构造”方法的可操作性，人们进一步提出了基于“软件机能构造”的开发体系和局部设计模型．基于此，把对象群作为角色、将各种角色进行拼装组合的技术，目前正越来越受到软件工程研究者的关注。

2.2.2 软件工程过程

方法和技术需要依据一定的流程、规范、衡量标准来进行，方可保证软件工程实施的有效性。软件工程过程就是这样一种规范，它使得软件能够被合理、及时地开发出来。软件工程过程定义了一组关键过程域的框架，这对于软件工程技术的有效应用是必须的。关键过程域构成了软件项目管理控制的基础，并且确定了上下各区域之间的关系，规定了技术方法的采用、工程产品（模型、文档、数据、报告、表格等）的产生、里程碑的建立、质量的保证及变更的适当管理。

美国卡内基・梅隆大学的研究成果 CMM/CMMI,为软件工程过程的评估和定义提供了参考模型(参见第 3 章)。CMM/CMMI 将软件开发组织的软件过程能力成熟度划分为多个级别,每个级别规定了关键的过程域,每个过程域又规定了相应的实践和特性,以此来评估软件开发组织是否在该过程领域达到了一定的成熟度。如果软件开发组织在某个级别的关键过程域均已达到成熟度水平,则认为其具有了该级别的软件过程能力。

CMM/CMMI 是一个评估模型,因此其在很大程度上还是在告诉软件开发组织"做什么"。至于"怎么做",CMM/CMMI 涉及不多。IBM 统一软件过程 RUP(IBM Rational Unified Process)则提供了一个更具有可操作性的实践框架,为软件过程定义提供了一个行之有效的工具。

RUP 中的软件生命周期在时间上被分解为四个顺序的阶段,分别是:初始阶段(inception)、细化阶段(elaboration)、构造阶段(construction)和交付阶段(transition)。每个阶段结束于一个主要的里程碑(major milestones);每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足。如果评估结果令人满意的话,可以允许项目进入下一个阶段。RUP 可以用二维坐标来描述。横轴通过时间组织,是过程展开的生命周期特征,体现开发过程的动态结构,用来描述它的术语主要包括周期(cycle)、阶段(phase)、迭代(iteration)和里程碑(milestone);纵轴以内容来组织为自然的逻辑活动,体现开发过程的静态结构,用来描述它的术语主要包括活动(activity)、产物(artifact)、工作者(worker)和工作流(workflow)(如图 2-1 所示)。

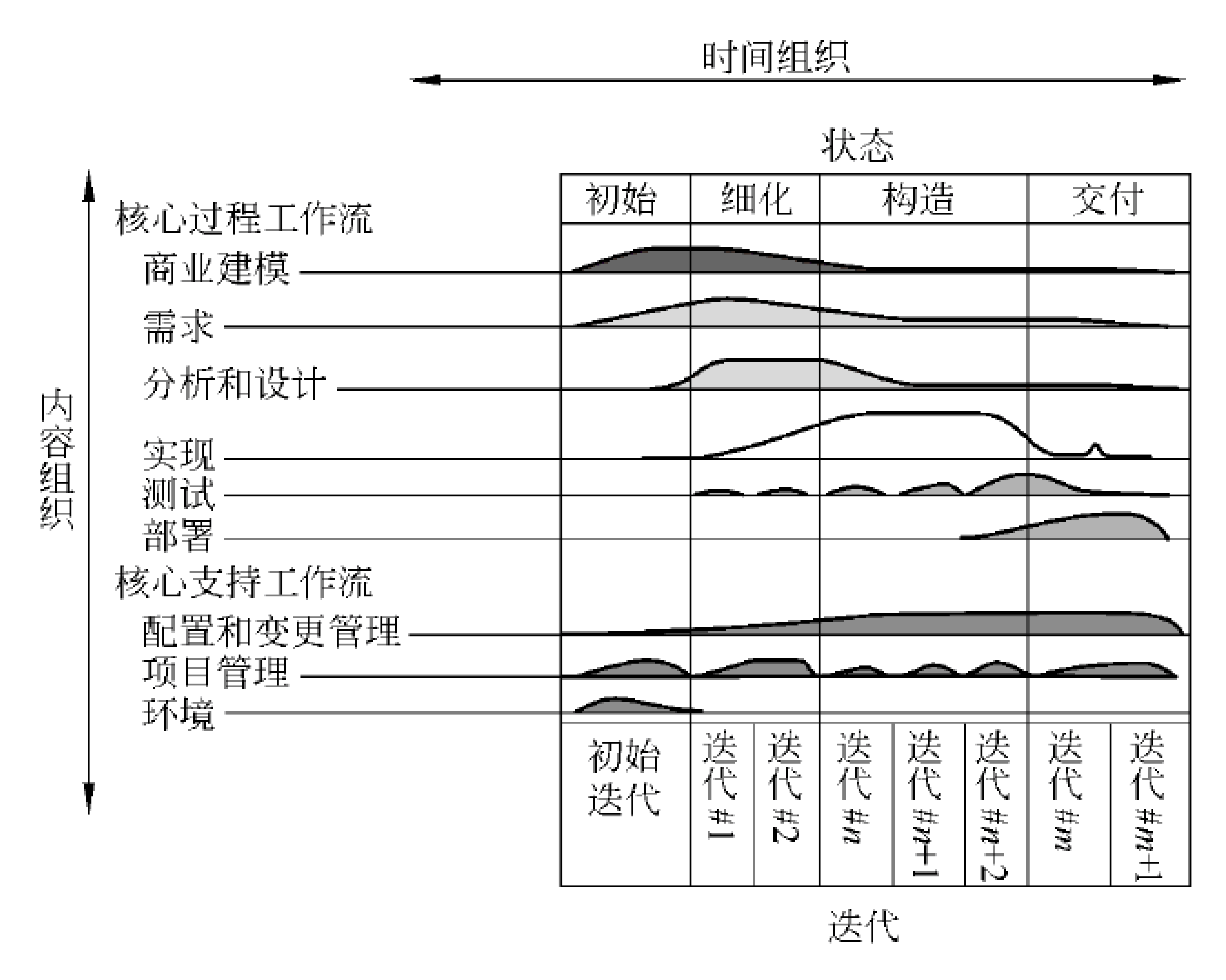


图 2-1　RUP 的软件过程

下面介绍 RUP 的各个阶段。

① 初始阶段。该阶段的目标是进行商业建模并确定项目的范围。为了达到该目的必须识别所有与系统交互的外部实体,在较高层次上定义交互的特性。

② 细化阶段。该阶段的目标是分析问题领域,建立对于软件体系结构、项目计划、项目范围、主要功能及性能的共同理解。同时为项目建立支持环境,包括创建开发案例,创

建模板、准则并准备工具。

③ 构造阶段。该阶段的主要目标是开发并集成产品，并进行软件测试。

④ 交付阶段。该阶段的主要目标是向客户交付产品。

RUP 中的每个阶段可以进一步分解为迭代。一个迭代是一个完整的开发循环，产生一个可执行的产品版本，是最终产品的一个子集，它增量式地发展，从一个迭代过程到另一个迭代过程到成为最终的系统。

2.2.3 软件工程工具

软件工程在一定的方法的指导下，依据某种过程进行，但也需要一定的辅助的软件工程工具。目前已经推出很多软件开发工具，如需求分析阶段的系统建模工具；编码阶段的各种语言编译工具、编辑程序、连接程序等；测试阶段的测试数据产生程序、动态分析程序、静态分析程序等软件自动测试工具；维护阶段的版本控制系统等。从广义上来讲，软件分析、设计阶段的各种图形工具，如数据流图（Data Flow Diagram，DFD）等也可以称为软件开发工具。

软件工程工具可以分为以下几种类型：

（1）管理工具。主要用于对软件项目管理活动进行支持，如项目管理工具 Microsoft Project、需求管理工具 Doors、测试管理工具 TestDirector、配置管理工具 Microsoft Visual SourceSafe 等。

（2）开发工具。主要对于软件的系统分析与设计、开发及测试活动进行支持，如系统建模工具 Rose、集成开发环境 JBuilder、Microsoft .NET Framework、单元测试工具 JUnit、性能测试工具 LoadRunner 等。

（3）其他工具。其他图形化工具，如用例图、鱼骨图等。

使用软件工程工具，可以在不同程度上简化软件开发和管理工作，提高工作效率，并且能够确保软件工程方法、软件工程过程的正确和有效。在某些阶段，软件工程工具是必需的，如在软件实现阶段，需要使用相应的集成开发环境。

2.2.4 CASE 简介

CASE（Computer Aided Software Engineering，计算机辅助软件工程），指在软件工程活动中，软件开发人员按照软件工程的方法和原则，借助于计算机及其软件工具的帮助来开发、维护和管理软件产品的过程。CASE 是一组工具和方法的集合，可以辅助软件开发生命周期各阶段进行软件开发，同时也是对方法的补充和替代，有助于生成高质量的软件。CASE 应支持软件工程各阶段；支持技术型软件工程；适应特定的阶段、特定的方法；应尽量使软件工程师的劳动自动化。

CASE 工具是指支持 CASE 的工具，如编译器、分析设计工具、测试工具等大部分软件工程工具。将 CASE 工具、信息按统一标准和接口组装起来，使工具间、人员间、各个过程间能方便交互，就形成了集成 CASE 环境。集成 CASE 环境对目前的软件开发来说必不可少。例如，Microsoft .NET Framework 将编辑、编译、调试、界面设计、安装程序生

成、配置管理、自动化测试等集成在一起,为整个项目的开发和管理提供了统一的平台。

2.3 软件工程的基本原则

围绕工程设计、工程支持以及工程管理提出了以下四条基本原则。

1. 选取适宜的开发模型

该原则与系统设计有关。在系统设计中,软件需求、硬件需求以及其他因素间是相互制约和影响的,经常需要权衡。因此,必须认识需求定义的易变性,采用适当的开发模型,保证软件产品满足用户的要求。

2. 采用合适的设计方法

在软件设计中,通常需要考虑软件的模块化、抽象与信息隐蔽、局部化、一致性以及适应性等特征。合适的设计方法有助于这些特征的实现,以达到软件工程的目标。

3. 提供高质量的工程支撑

工欲善其事,必先利其器。在软件工程中,软件工具与环境对软件过程的支持颇为重要。软件工程项目的质量与开销直接取决于对软件工程所提供的支撑质量和效用。

4. 重视软件工程的管理

软件工程的管理直接影响可用资源的有效利用,生产满足目标的软件产品以及提高软件组织的生产能力等问题。因此,仅当软件过程予以有效管理时,才能实现有效的软件工程。

2.4 软件工程的原理

自从 1968 年出现“软件工程”这个术语以来,专家学者们陆续提出了 100 多条关于软件工程的准则或“信条”,希望能对软件工程活动提供指导和建议。著名的软件工程学家 B. W. Boehm 综合这些学者们的意见并总结了 TRW 公司多年开发软件的经验,于 1983 年在一篇论文中提出了软件工程的 7 条基本原理。他认为这 7 条原理是确保软件产品质量和开发效率原理的最小集合,每一条原理既互相独立,又不可或缺。然而这 7 条原理又是完备的,在此之前已经提出的 100 多条软件工程原理都可以由这 7 条原理的任意组合蕴含或派生。

1. 用分阶段的生存周期计划严格管理

把软件生命周期划分成若干个阶段,并相应地制定出切实可行的计划,然后严格按照计划对软件的开发与维护工作进行管理。Boehm 认为,在软件的整个生命周期中应该制定并严格执行六类计划,它们是项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划。

2. 坚持进行阶段评审

大部分错误是在编码之前造成的,例如,根据 Boehm 等人的统计,设计错误占软件错

误的 63%，编码错误仅占 37%；错误发现与改正得越晚，所需付出的代价也越高。因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的错误。

3. 实行严格的产品控制

在软件开发过程中遭遇需求变化是不可避免的。为了保持软件各个配置成分的一致性，必须实行严格的产品控制，其中主要是实行基线配置管理。所谓基线配置，是指经过阶段评审后的软件配置成分(各个阶段产生的文档或程序代码)。基线配置管理也称为变更控制：一切有关修改软件的建议，特别是涉及对基线配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。

4. 采用现代程序设计技术

从提出软件工程的概念开始，人们一直把主要精力用于研究各种新的程序设计方法与技术。20 世纪 60 年代末提出的结构化程序设计方法，已经成为绝大多数人公认的先进的程序设计技术。近年来，面向对象技术已经在许多领域中迅速地取代了传统的结构化开发方法。实践表明，采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量。

5. 结果应能清楚地审查

软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员(或开发小组)的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评价和管理。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查。

6. 开发小组的人员应少而精

软件开发小组的组成人员的素质应该好，而人数则不宜过多。开发小组人员的素质和数量是影响软件产品质量和开发效率的重要因素。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为 N 时，可能的通信路径有 $N(N-1)/2$ 条，可见随着人数 N 的增大，通信开销将急剧增加。因此，组成少而精的开发小组是软件工程的一条基本原理。

7. 承认不断改进软件工程实践的必要性

遵循上述六条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产，但是，仅有上述六条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步。因此，Boehm 提出应把承认不断改进软件工程实践的必要性作为软件工程的第七条基本原理。按照这条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验，例如，收集进度和资源耗费数据，收集出错类型和问题报告数据等。这些数据不仅可以用来评价新的软件技术的效果，而且可以用来指明必须着重开发的软件工具和应该优先研究的技术。

2.5 软件开发过程模型

世间万事万物都有一个从诞生到覆灭的过程，软件亦是如此。软件的生存周期是指一个计算机软件从功能确定、设计，到开发成功投入使用，并在使用中不断地修改、增补和完善，直到停止该软件的使用的全过程。一般来说，软件生存周期包括计划、开发、运行三个时期，每一时期又可分为若干更小的阶段。计划时期的主要任务是分析用户要求，分析新系统的主要目标以及开发该系统的可行性。开发时期要完成设计和实现两大任务。运行时期是软件生存周期的最后一个时期，软件人员在这一时期的工作，主要是做好软件维护。软件生存周期的划分，有助于对软件项目进行更合理的管理。

软件开发过程模型是软件开发全过程、软件开发活动以及它们之间关系的结构框架，它为软件项目的管理提供里程碑和进度表，为软件开发提供原则和方法。可以说软件开发过程模型是贯穿于整个软件生存周期的，在其各个阶段发挥重要的作用。

2.5.1 瀑布模型

瀑布模型是最早的软件开发方法之一，它将软件生存周期的各项活动规定为按固定顺序连接的若干阶段工作，形如瀑布流水，最终得到软件产品（如图 2-2 所示）。其特点是：阶段间的顺序性和依赖性，上一阶段结束才能进入下一阶段；每一阶段以前一阶段的结果为基础；要求软件需求阶段十分完善。

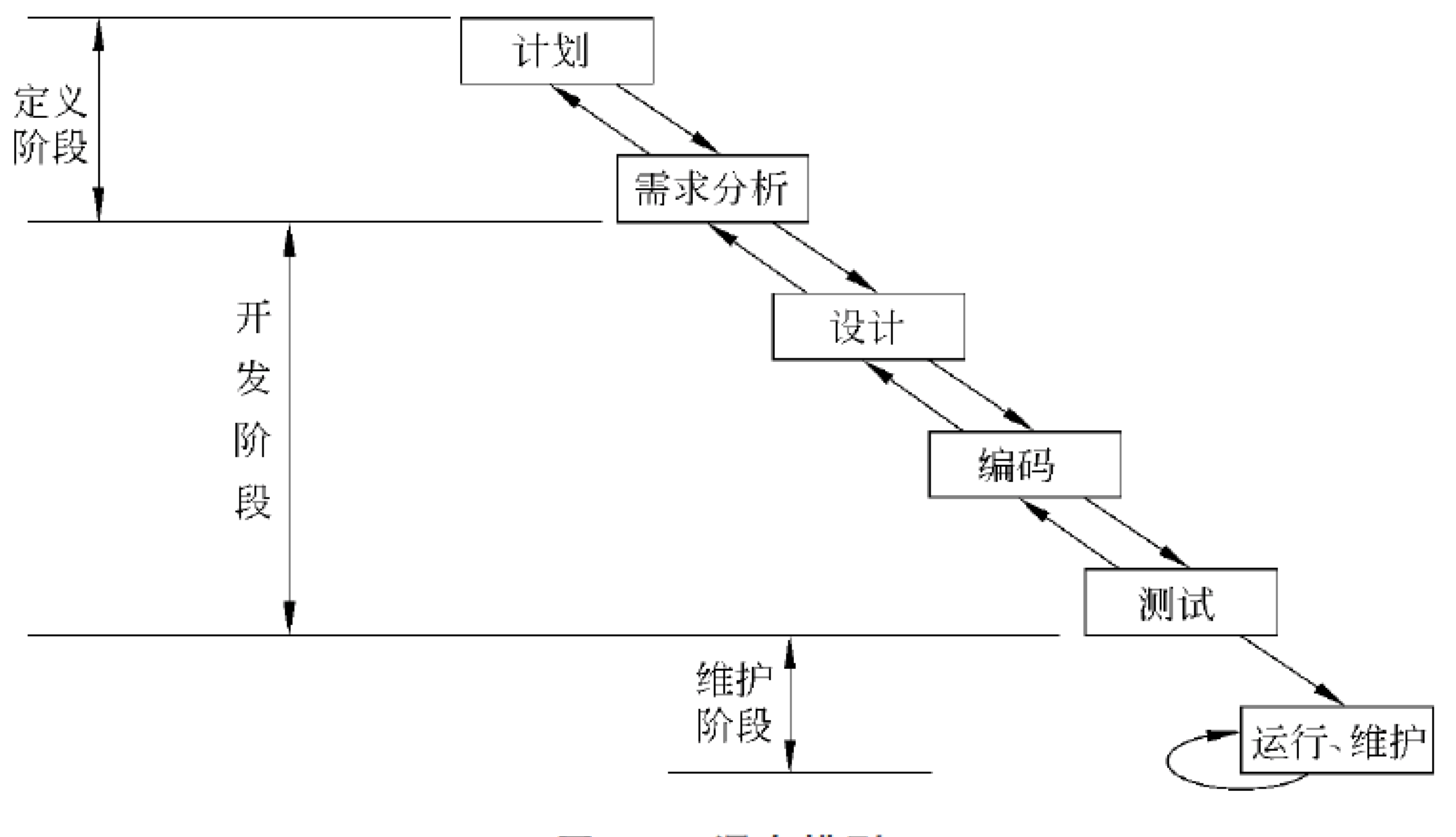


图 2-2　瀑布模型

这种模型的弊端在于，当在某一阶段发现错误时，需要回溯到产生错误的源头阶段，然后从源头阶段重新进行相应的生存周期活动，既而向下一阶段进行。如果错误发现得越晚，那么修改的成本可能越大。因此，这种模型适合于规模较小，且需求相对明确和固定的软件项目。虽然如此，瀑布模型仍然可以作为其他软件开发过程模型的鼻祖，因为阶段的划分是其他模型共有的特点。

2.5.2 原型模型

原型模型主要针对事先不能完整定义需求的软件项目。软件开发人员根据用户提出的基本软件需求,借助开发工具尽快地构造一个实际系统的简化模型,作为系统的框架,便于开发者与用户之间进行交流。根据用户对系统的意见反馈使需求进一步精确化、完全化,并据此改进、完善原型,如此反复迭代,直到软件开发人员和用户都确认软件系统的需求并达成一致的理解为止(如图 2-3 所示)。

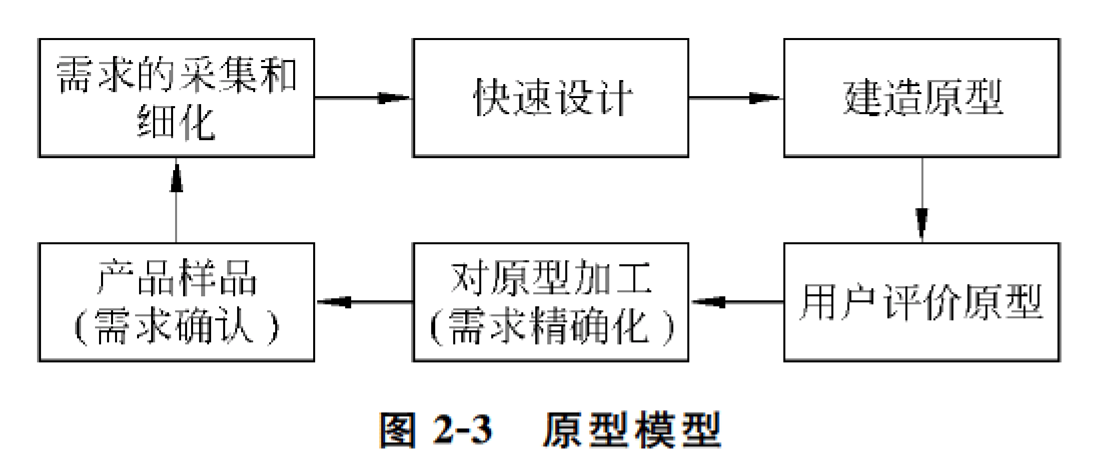


图 2-3　原型模型

从严格意义上来讲,原型模型只是在需求定义阶段使用的一种模型,开发出来的原型系统也只是用于与客户进行需求确认之用,在需求确认之后,原型系统就要被废弃,然后重新进行软件系统的开发。

2.5.3 增量模型

增量模型是瀑布模型和原型模型的结合,强调版本升级,每个版本的开发遵循一定的顺序。增量模型对软件开发活动进行如下组织:在设计了软件系统整体体系结构之后,一般先完成一个系统子集的开发,再按同样的开发步骤增加功能(系统子集),如此递增下去直至满足全部系统需求(如图 2-4 所示)。

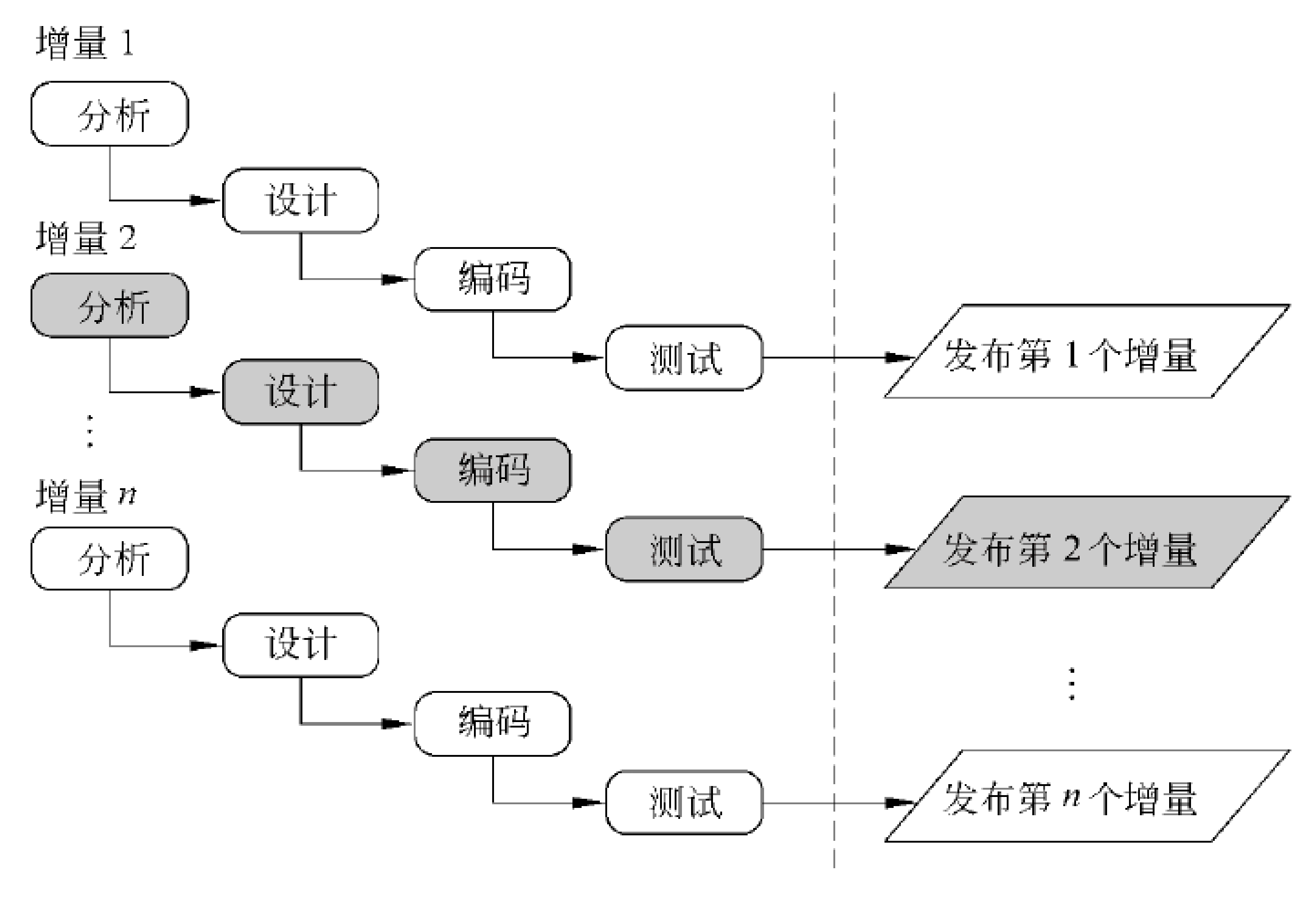


图 2-4　增量模型

增量模型一般适用于规模较大，子系统较多，且难以一次性明确全部需求的软件项目。增量模型的核心思想是把软件产品分解成一系列的增量构件，在增量开发迭代中逐步加入。每个构件由多个相互作用的模块构成，并且能够完成特定的功能，满足指定的用户需求（如图 2-5 所示）。增量开发方法有一个最新演进版本，就是“极限编程（eXtreme Programming，XP）”。XP 编程在软件开发领域有着众多的使用者。

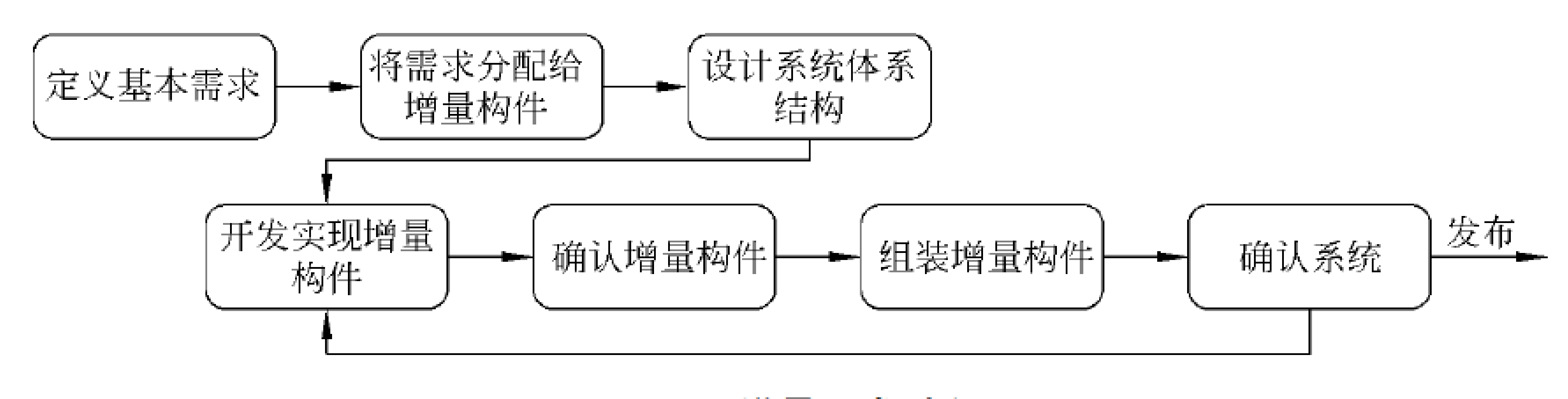


图 2-5　增量开发过程

2.5.4　喷泉模型

喷泉模型是近几年提出来的软件生存周期模型。它是以面向对象的软件开发方法为基础，以用户需求为动力，以对象来驱动的模型，是典型的面向对象生命周期模型。该模型认为软件生命周期的各个阶段是多次重复和重叠的。

喷泉模型体现了面向对象软件开发过程迭代和无缝的特性，为避免喷泉模型开发软件时开发过分无序，应该把一个线性过程作为总目标。面向对象范型本身要求经常对开发活动进行迭代或求精。喷泉模型又具有增量开发特性，即能做到分析一点、设计一点、实现一点，测试一点，使相关功能随之加入到演化的系统中（如图 2-6 所示）。

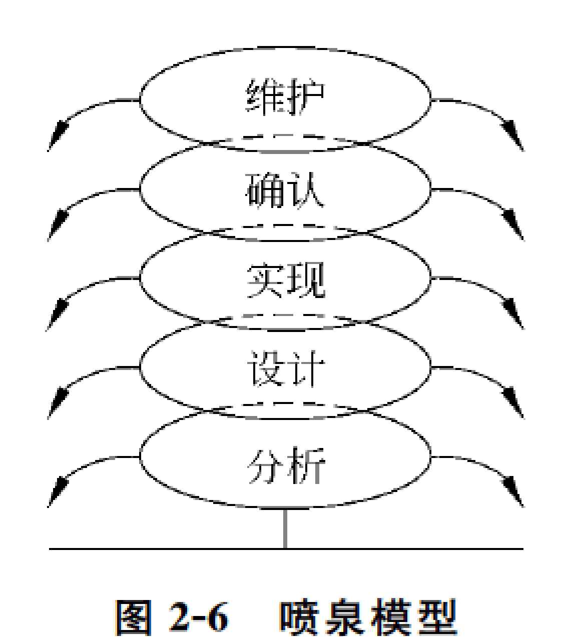


图 2-6　喷泉模型

2.5.5　螺旋模型

螺旋模型在瀑布模型和原型模型基础上逐渐修正，引入了风险分析，遵循需求→架构→设计→开发→测试的路线。同时整个开发过程又是迭代和风险驱动的，通过将瀑布模型的多个阶段转化到多个迭代过程中，以减少项目的风险（如图 2-7 所示）。

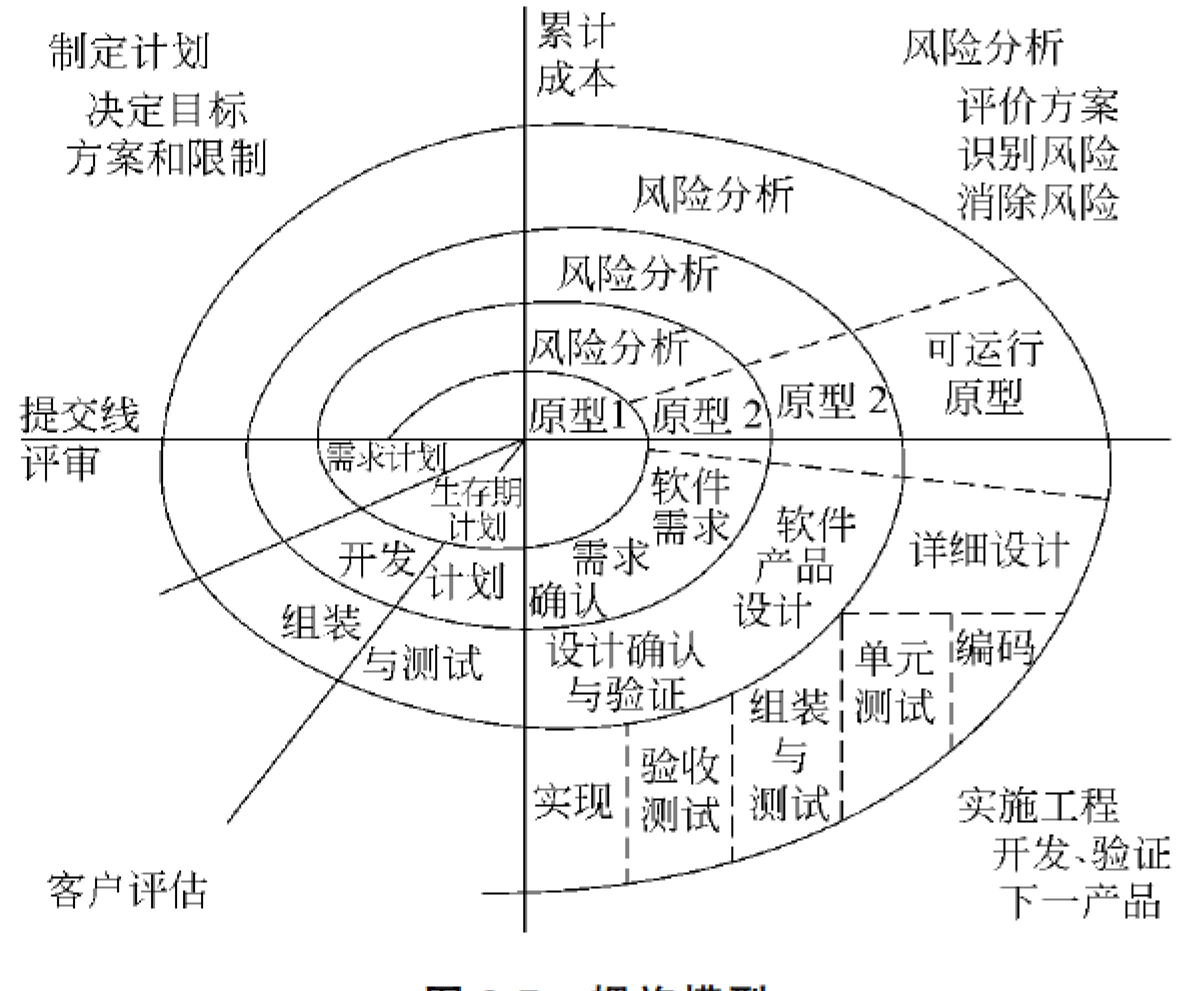


图 2-7　螺旋模型

2.6　软件工程标准

2.6.1　标准概述

什么是标准？为什么要使用标准？

标准是对重复性事物和概念所作的统一规定。它以科学技术和实践经验的综合成果为基础，按照法定程序，经过参与方协商一致，由某个公认机构批准、发布，作为相关各方共同遵守的准则和依据。

标准是经验的总结。人们在工作和生产的过程中，不断地摸索和总结经验教训，逐渐发现某些活动可以按照既定的流程、方法、原则来开展，于是在摒弃了大量的错误、不足之后，形成了能够反映正确、合理、科学的工作和生产规律的一些原理。将这些原理推而广之，便逐渐形成了标准。因此，标准是实践经验的优秀总结。

标准就是规范化。标准化的过程是在经济、技术、科学和管理等社会实践中，通过制定、实施标准达到统一，以获得最佳秩序和社会效益的过程。标准化的过程实质就是规范化的过程。

标准是沟通的基础。有了标准，人们就可以在共同的基础上进行沟通，从而可以防止出现信息的误解和丢失。

2.6.2　标准分类

根据标准的制定机构和标准的适用范围，可以将标准分为国际标准、国家标准、地方标准、行业标准、企业（机构）标准及项目标准等几大类。

我国标准分为国家标准、行业标准、地方标准和企业标准四级。对需要在全国范畴内

统一的技术要求，应当制定国家标准。对没有国家标准而又需要在全国某个行业范围内统一的技术要求，可以制定行业标准。对没有国家标准和行业标准而又需要在省、自治区、直辖市范围内统一的工业产品的安全、卫生要求，可以制定地方标准。企业生产的产品没有国家标准、行业标准和地方标准的，应当制定相应的企业标准。对已有国家标准、行业标准或地方标准的，鼓励企业制定严于国家标准、行业标准或地方标准要求的企业标准。

另外，对于技术尚在发展中，需要有相应的标准文件引导其发展或具有标准化价值，尚不能制定为标准的项目，以及采用国际标准化组织、国际电工委员会及其他国际组织的技术报告的项目，可以制定国家标准化指导性技术文件。

1. 国际标准

国际标准是指由国际联合机构制定和公布，提供各国参考的标准。目前国际上最有影响的国际标准制定机构有 ISO(International Standard Organization，国际标准化组织)和 IEC(International Electro-technical，国际电工委员会)。

ISO 成立于 1947 年 2 月 23 日，是世界上最大的国际标准化组织，其前身是 1928 年成立的国际标准化协会国际联合会(International Standards Association，ISA)。ISO 的宗旨是"在世界上促进标准化及其相关活动的发展，以便商品和服务的国际交换，在智力、科学、技术和经济领域开展合作"。ISO 现有 138 个成员，包括 138 个国家和地区，其最高权力机构是每年一次的"全体大会"。ISO 制定标准的过程具有开放性，并建立了力图吸引一切有兴趣参与者的透明程序，有强有力的解决分歧的能力。ISO 标准在国际上拥有较为广泛的认同，每 5 年需要对所制定的标准进行审查以决定是否需要肯定、修订、废弃。

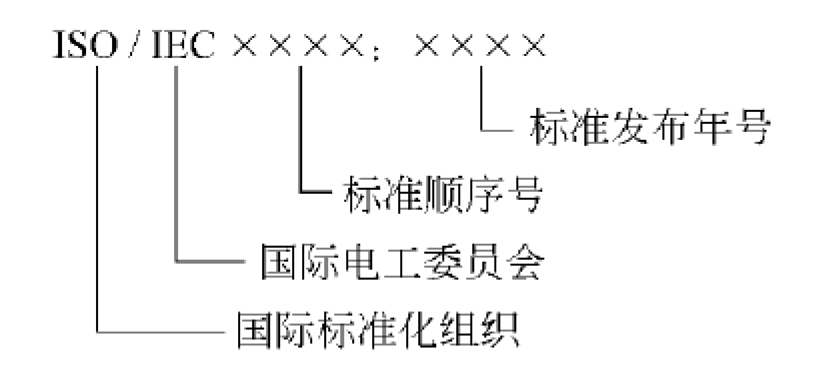
IEC 成立于 1906 年，是世界上最早的国际标准化组织。IEC 主要负责电工、电子领域的标准化活动。而 ISO 负责除电工、电子领域之外的所有其他领域的标准化活动。20 世纪 90 年代初，ISO 和 IEC 进行合作，成立了联合技术委员会(Joint Technical Committee1，JTC1)，致力于信息技术标准化。因此，现在经常会看到以 ISO/IEC 开头的信息技术类国际标准。

TR(Technical Report)是 ISO/IEC 发布的技术报告，如 ISO/IEC TR 15504-1 发布技术报告一般需满足以下条件：

① 在一个技术委员会内，当成员团体将建议草案作为国际标准草案呈报、或对国际标准草案进行表决时，虽然经过多次努力，仍未能获得必要的多数票(或实质性的支持)，则该技术委员会可申请以技术报告的形式来发表一份文件，并且在该文件中应注明未能获得必要多数票的原因。

② 当有关项目还处于技术发展过程中，以及根据其他理由认为今后某个时候有达成协议的可能，则技术委员会可申请发表一份较恰当的技术报告。

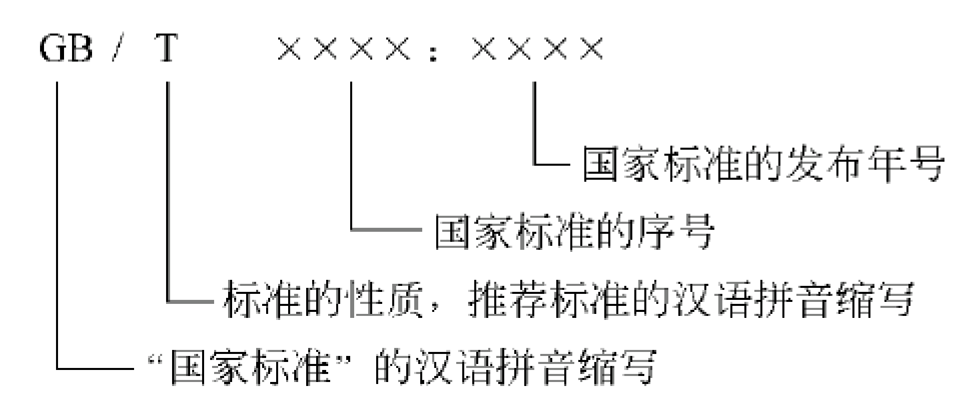
③ 当某个技术委员会汇集了一些与国际标准性质不同的资料(例如包括对成员团体进行调查后获得的真实情况，有关其他国际组织内的工作情况，或某些成员团体有关的标准的某一特定项目"目前工作水平"的情报)，技术委员会可以技术报告形式出版这些资料。



2. 国家标准

国家标准由政府或国家级的机构制定或批准，适用于全国范围。中华人民共和国国家质量技术监督局是我国最高标准化机构，负责组织制定和颁布我国的有关标准。另外，如美国国家标准协会（America National Standards Institute，ANSI）、英国国家标准（British Standard，BS）、日本工业标准（Japanese Industrial Standard，JIS）分别是美国、英国、日本等国相应的国家标准制定机构。

我国标准的编号由标准代号、标准发布顺序和标准发布年代号构成。国家标准的代号由"国标"这两字的大写汉语拼音字母构成，GB 为强制性国家标准代号，GB/T 为推荐性国家标准的代号。



3. 行业标准

行业标准由行业机构、学术团体或国防机构制定，适用于某个业务领域。相关的标准化制定机构如美国电气与电子工程师学会（Institute of Electrical and Electronics Engineers，IEEE），其前身是成立于 1884 年的 AIEE（美国电气工程师协会）和 IRE（无线电工程师协会），1963 年 1 月 1 日 AIEE 和 IRE 正式合并为 IEEE。自成立以来 IEEE 一直致力于推动电工技术在理论方面的发展和应用方面的进步。IEEE 是一个非营利性科技学会，拥有全球近 175 个国家 360 000 多名会员。透过多元化的会员，该组织在太空、计算机、电信、生物医学、电力及消费性电子产品等领域中都是主要的权威。在电气及电子工程、计算机及控制技术领域中，IEEE 发表的文献占了全球将近 30%。IEEE 每年也会主办或协办 300 多项技术会议。IEEE 计算机协会的软件工程标准委员会一直从事软件工程标准的制订，发布了大量软件工程的标准，对各国的软件工程标准产生了重大影响。IEEE 通过的标准通常要报请 ANSI 审批，使其具有国家标准的性质。因此，我们常可以看到 IEEE 公布的标准会含有 ANSI 字样。

在我国，行业标准代号由有汉语拼音大写字母组成，再加上斜线和 T 组成推荐性行业标准，如××/T。行业标准代号由国务院各有关行政主管部门提出其所管理的行业标准范围的申请报告，国务院标准化行政主管部门审查确定并正式公布该行业标准代号，我国已经正式发布的行业代号有 QJ（航天）、SJ（电子）、JB（金融系统）等。典型的行业标准还有中华人民共和国国家军用标准，其标准代号为 GJB，这是由我国国防科学技术工业委员会批准，适合于国防部门和军队使用的标准。

4. 地方标准

地方标准又称为区域标准：对没有国家标准和行业标准而又需要在省、自治区、直辖市范围内统一的工业产品的安全、卫生要求，可以制定地方标准。我国的地方标准由省、自治区、直辖市标准化行政主管部门制定后，报国务院标准化行政主管部门和国务院有关行政主管部门备案。地方标准在公布相关国家标准或者行业标准之后，即应自动废止。

地方标准代号由大写汉语拼音 DB 加上省、自治区、直辖市行政区划代码的前面两位数字(如北京市 11、天津市 12、上海市 13 等)，再加上斜线和 T 组成推荐性地方标准(DB××/T)，不加斜线 T 为强制性地方标准(DB××)。

5. 企业标准

企业标准，或称企业规范，是由一些大型企业或公司出于软件工程工作的需要，制定的适用于本公司或部门的规范，仅在公司内部使用。在我国，企业标准的代号由汉字大写拼音字母 Q 加斜线再加企业代号组成(Q/×××)，企业代号可用大写拼音字母或阿拉伯数字或者两者兼用所组成。

2.6.3 软件工程相关标准介绍

标准是对重复性事物和概念所作的统一规定，它是人们从实践和探索中提炼出来的，具有普遍的适用性，对于本领域的发展起到了莫大的作用，同时也促进了国际、国内间的交流和知识共享。因此，从事软件工程这个行业，积极地了解、掌握和应用这些成熟的标准无论对于自身的提高还是所属组织的发展都是大有裨益的。

本节试图从软件工程的角度，向大家介绍一些常用的软件工程标准，这其中既有软件开发过程域的，也有本书将重点讲述的软件测试过程域的，因为软件工程本质上就是各个过程域工程化后相辅相成的软件构建过程，软件测试只是其中一个领域。但是，要想彻悟软件测试，还是应该要回归到软件工程这个中心，任何在软件测试领域的尝试都是在为软件工程服务。

1. 软件开发过程标准

1) GB/T 8566—2007《信息技术 软件生存周期过程》

软件生存周期是指软件从构思开始至软件退役为止的软件发生、发展直至软件退役(死亡)的整个生存周期。为了开发高水平、高质量的软件(特别是大型软件)，提升软件应用的效率，软件的开发和维护需要通过一个科学的过程来控制和管理。随着软件产业的兴起，许多专家对软件过程管理与控制做了大量和深入的研究，在此基础上，IEEE 和 ISO 总结归纳了这些研究成果，经过不断的讨论与修改，逐步形成了过程标准，并于 1995 年正式推出了国际标准 ISO/IEC 12207：1995《信息技术 软件生存周期过程》(注：为方便阅读，本书所涉及的国际标准统一使用中文表述)。

我国于 2007 年正式颁布的 GB/T 8566—2007 综合了 ISO/IEC12207：1995《信息技术 软件生存周期过程》和 ISO/IEC12207 Amd. 1：2002、ISO/IEC12207Amd. 2：2004 的有关内容，并予以修改采用。

GB/T 8566—2007 标准将软件生存周期过程分为生存周期基本过程、生存周期支持

过程、生存周期组织过程三个类别(如图 2-8 所示)。基本过程是针对不同的使用者而规定获取、开发、维护软件需要开展的活动及任务;支持过程是规定为支持实施基本过程而需要开展的活动及任务;组织过程是规定为支持实施基本过程和支持过程而在组织层面需要开展的活动及任务。

生存周期基本过程	生存周期支持过程	生存周期组织过程
获取过程	文档编制过程	管理过程
供应过程	配置管理过程	基础设施过程
开发过程	质量保证过程	改进过程
运作过程	验证过程	人力资源过程
维护过程	确认过程	资产管理过程
	联合评审过程	重用大纲管理过程
	审核过程	领域工程管理过程
	问题解决过程	
	易用性过程	

图 2-8　GB/T 8566 中的软件生存周期过程

(1) 生存周期基本过程包括 5 个过程,这些过程供各主要参与方在软件生存周期期间使用。主要参与方是参与或完成软件产品开发、运作或维护的组织。这些主要参与方有软件产品的需方、供方、开发方、操作方和维护方。

基本过程是:

① 获取过程。为获取系统、软件产品或软件服务的组织即需方而定义的活动。

② 供应过程。为向需方提供系统、软件产品或软件服务的组织即供方而定义的活动。

③ 开发过程。为定义并开发软件产品的组织即开发方而定义的活动。

④ 运作过程。为在规定的环境中为其用户提供运行计算机系统服务的组织即操作方而定义的活动。

⑤ 维护过程。为提供维护软件产品服务的组织即维护方而定义的活动。也就是对软件的修改进行管理,使它保持合适的运行状态。该过程包括软件产品的迁移和退役。

生存周期支持过程包括 9 个过程。支持过程以明确的目的作为构成整体所必须的部分支持其他过程(主要是基本过程)。有助于软件项目的成功和提高质量。支持过程按照其他过程的需要采用和执行。支持过程有:

① 文档编制过程。为记录生存周期过程所产生的信息而定义的活动。

② 配置管理过程。定义配置管理活动。

③ 质量保证过程。为客观地保证软件产品和过程符合规定的需求以及已建立的计划而定义的活动。联合评审、审核、验证和确认可以作为质量保证技术使用。

④ 验证过程。根据软件项目需求,按不同深度(为需方、供方或某独立方)验证软件产品而定义的活动。

⑤ 确认过程。(为需方、供方或某独立方)确认软件项目的软件产品而定义的活动。

⑥ 联合评审过程。为评价一项活动的状态和产品而定义的活动。该过程可由任何

两方应用，其中一方（评审方）以联合讨论会的形式评审另一方（被评审方）。

⑦ 审核过程。为判定符合需求、计划和合同而定义的活动。该过程可由任何两方应用，其中一方（审核方）审核另一方（被审核方）的软件产品或活动。

⑧ 问题解决过程。为分析和解决问题（包括不合格）而定义的活动，不论问题的性质或来源如何，它们都是在实施开发、运作、维护或其他过程期间暴露出来的。

⑨ 易用性过程。为易用性专业人员而定义的活动。

（2）生存周期组织过程包括 7 个过程。这些过程可被某个组织用来建立和实现由相关的生存期过程和人员组成的基础结构并不断改进这种结构和过程。采用它们通常超出特定的项目和合同的范围。但是，这些特定项目和合同的经验教训有助于改善组织状况。

① 管理过程。为生存周期过程中的管理包括项目管理而定义的基本活动。

② 基础设施过程。为建立生存周期过程基础设施而定义的基本活动。

③ 改进过程。为某一组织（即需方、供方、开发方、操作方、维护方或另一过程的管理者）建立、测量、控制和改进其生存周期过程而定义需要执行的基本活动。

④ 人力资源过程。为给组织或项目拥有技能和知识的员工而定义的活动。

⑤ 资产管理过程。为组织的资产管理者而定义的活动。

⑥ 重用大纲管理过程。为组织的重用大纲主管而定义的活动。

⑦ 领域工程管理过程。为领域模型、领域体系结构的确定及该领域资产的开发和维护而定义的活动。

2）GB/T 8567—2006《计算机软件文档编制规范》

在现代大型软件的开发过程中，软件文档的重要性不言而喻，它是软件开发团队中成员主要的沟通媒介，提高了软件开发的透明度，使开发进程更加容易被度量和控制。

GB/T 8567—2006《计算机软件文档编制规范》就是专门针对文档编制而发布的国家标准，通过标准对软件的开发过程和管理过程应编制的主要文档及其编制的内容、格式做了基本要求。GB/T 8567—2006《计算机软件文档编制规范》原则上适用于所有类型的软件产品的开发过程和管理过程。

GB/T 8567—2006《计算机软件文档编制规范》中给出了一个文档编制规范（如图 2-9 所示）。在这个规范中，我们可以看到标准的主要活动是建立开发文档的计划，这是必要的，因为有计划，文档编制的质量会更好，过程的效率会更高。

文档开发计划必须包括风格规格说明。本标准不规定风格规格说明的内容（即不规定具体的布局和字体），但对于每类文档必须覆盖什么则做出了相应规定，并且规定了何种信息对于文档管理者是可用的和谁做评审和再生产文档。

3）ISO/IEC 15504：2004《信息技术 过程评估》

1991 年，ISO/IEC 第一联合技术委员会的第七副委员会（JTCI/SC7）提出发展一套软体过程评估的国际标准的决议，1993 年，ISO/IEC JTCI 采用了这个决议，发起制订 ISO/IEC 15504 系列标准的前期工作，并将其命名为“软件过程改进和能力确定”（Software Process Improvement and Capability Determination，SPICE）项目。

1994 年，SPICE 项目的基准文件出台，同时，ISO/IEC JTCI 以基准文件为基础，在全球范围内展开大规模试验。在 SPICE 试验成功进行的基础上，1998 年，ISO/IEC JTCI

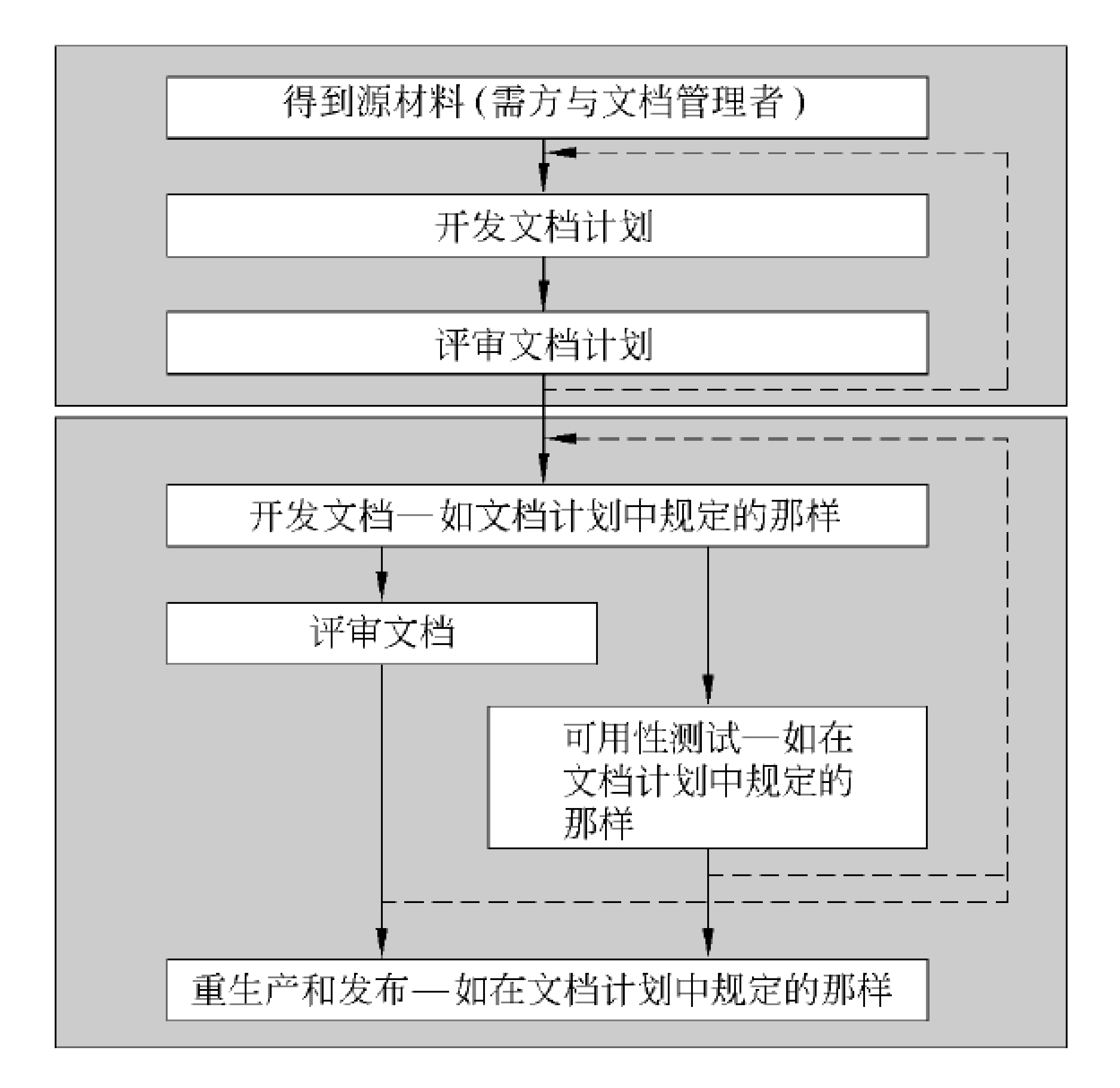


图 2-9　GB/T 8567 中的文档过程

正式发布了 ISO/IEC 15504 TR 系列技术报告，该技术报告由“概念和介绍性指南”、“过程和过程能力的参考模型”、“实施评估”、“实施和指标指南”、“过程评估模型”、“评估员资格指南”、“过程改进指南”、“确定供应者过程能力应用指南”、“词汇”9 部分组成。

1994 年至今，ISO/IEC JTCI 一直在世界各地大力推动 SPICE 试验，同时根据反馈信息，着手制定正式标准，到目前已形成 ISO/IEC 15504 标准，该标准是软件过程评估的国际标准，提供了一个软件过程评估的框架，可以被任何组织用于软件的设计、管理、监督、控制，以及提高“获得、供应、开发、操作、升级和支持”的能力。另外，该标准还定义了一种结构化的软件过程评估方法，旨在为描述工程评估结果的通用方法提供一个基本原则，同时也对建立在不同但兼容的模型和方法上的评估进行比较。

ISO/IEC 15504 标准由 5 部分组成，它们之间的关系如图 2-10 所示。

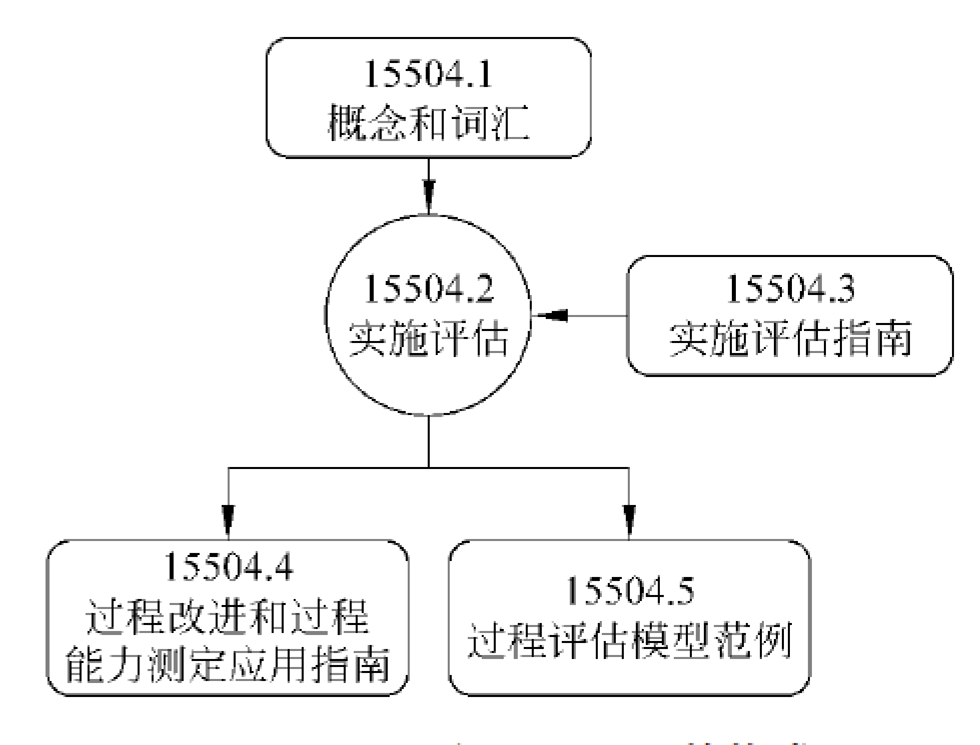


图 2-10　ISO/IEC 15504 的构成

(1) ISO/IEC 15504.1《信息技术 过程评估 第 1 部分：概念和词汇》是由 ISO/IEC TR 15504.1 和 ISO/IEC TR 15504.9 合并、修改形成，介绍与软件过程评估相关的概念和词汇。

(2) ISO/IEC 15504.2《信息技术 过程评估 第 2 部分：实施评估》是由 ISO/IEC TR 15504.3 修改形成。ISO/IEC 15504.2 定义实施过程要求，作为使用过程改进和能力测定的基础；确定过程能力测量框架和确定实施评估、过程参考模型、过程评估模型、验证过程评估一致性的要求；规定了要求的最小集合，使其能够保证评估结果的客观、公正、一致和可重复，保证被评估过程具有代表性。

(3) ISO/IEC 15504.3《信息技术 过程评估 第 3 部分：实施评估指南》是由 ISO/IEC TR 15504.4 和 ISO/IEC TR 15504.6 合并、修改形成。ISO/IEC 15504.3 提供指南以满足 ISO/IEC 15504.2 规定的、执行评估要求的最小集合，提供过程评估的总的看法，提供执行评估、过程能力测量框架、过程参考模型和过程评估模型、选择和应用评估工具、评审员资格、验证一致性的指南，解释相关要求；还提供了一个符合 ISO/IEC 15504.2 4.2 节要求的评估过程样本文件。

(4) ISO/IEC 15504.4《信息技术 过程评估 第 4 部分：过程改进和过程能力测定应用指南》是由 ISO/IEC TR 15504.4 和 ISO/IEC TR 15504.6 合并、修改形成。ISO/IEC 15504.4 为过程改进和过程能力测定提供指南，在一个过程改进(PI)的环境中，过程评估利用选择的过程和能力，提供了表征一个组织单元的方法。分析过程评估的结果，对照一个组织单元的业务目标可以识别这些过程的效力、弱点和风险。这个结果反过来有助于确定这些过程对实现企业目标是否有效并提供改进动力。ISO/IEC 15504.4 叙述了 PI 和 PCD(过程能力测定)，叙述了如何配置 PI 和 PCD。

(5) ISO/IEC 15504.5《信息技术 过程评估 第 5 部分：过程评估模型范例》是由 ISO/IEC TR 15504.2 和 ISO/IEC TR 15504.5 合并、修改形成。ISO/IEC 15504.5 的过程评估模型范例中，将对评估模型作出比 ISO/IEC TR 15504.2 更加详细和完备的叙述。其中评估模型的过程维，不再采用 ISO/IEC TR 15504.2 的描述和分类，而是由一个外部参考模型提供，简称为 PRM。

2. 度量与评价标准

对软件产品质量的全面说明和评价是保证足够质量的关键因素，而这可以通过在考虑软件产品的使用目的的情况下，定义适当的质量特性来实现。为了保证软件质量评价的公正性和认可度，规定和评价每个相关的软件产品质量特性时要尽可能使用经确认的或被广泛接受的度量(用于测量的一种量化的标度和方法)。本节将向大家介绍两个在软件测试领域广泛使用的软件评测标准：GB/T 16260—2006《软件工程 产品质量》和 GB/T 18905—2002《软件工程 产品评价》。

软件质量评价的基本部分包括质量模型、评价方法、软件的测量和支持工具。要想开发好的软件，宜规定质量需求，宜策划、实现和控制软件质量保证过程，宜评价中间产品和最终产品。要达到评价软件质量的目的，宜用有效的度量方法进行测量软件的质量属性。GB/T 16260—2006《软件工程 产品质量》系列标准定义了一个通用质量模型和质量特性，并给出了度量的示例；GB/T 18905—2002《软件工程 产品评价》系列标准给出软件产

品评价过程的概述，并且提供了用于评价的要求和指导。图 2-11 所示的是两个系列标准中各部分之间的关系。

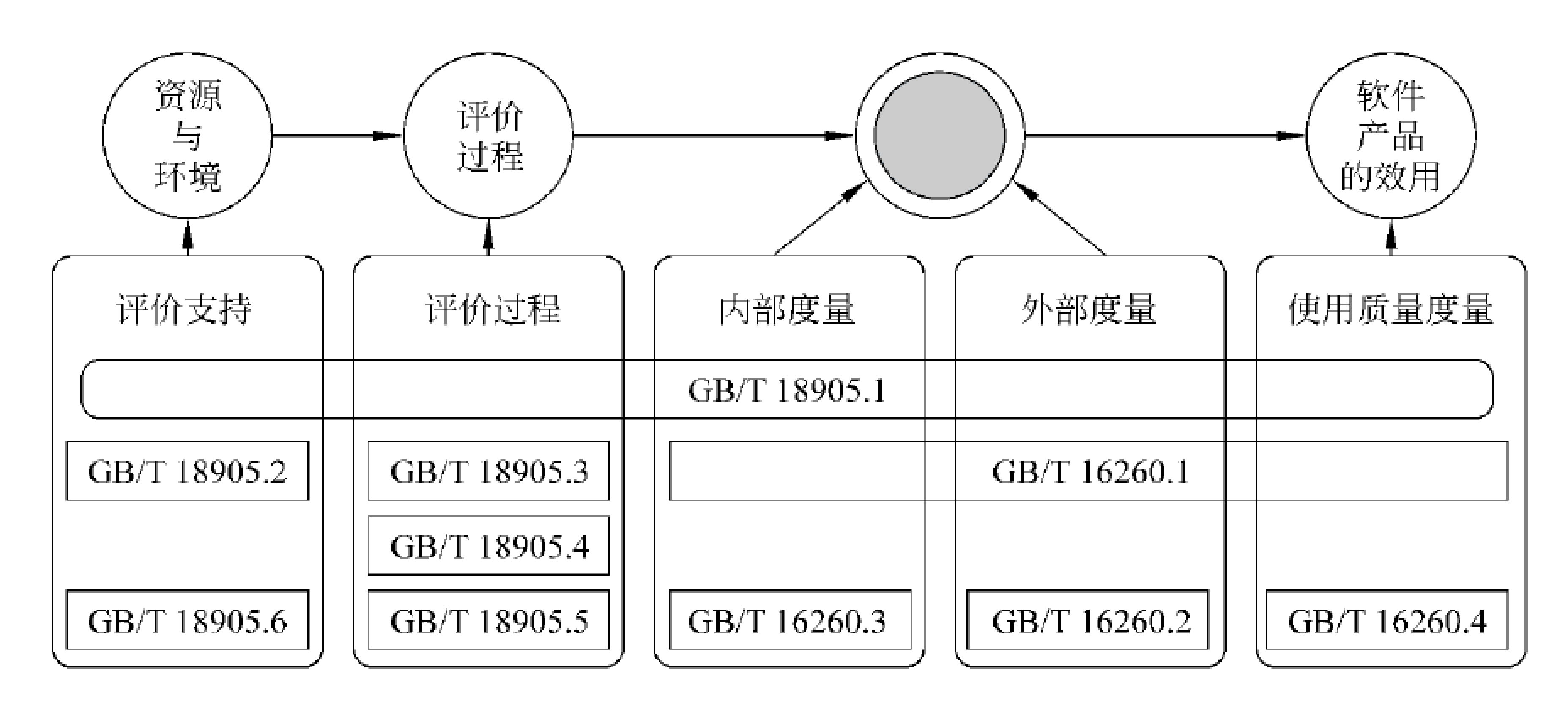


图 2-11　GB/T 16260 与 GB/T 18905 标准之间的关系

1）GB/T 16260—2006《软件工程 产品质量》

GB/T 16260—2006《软件工程 产品质量》是我国于 2006 年发布的国家标准，内容等同采用自 ISO/IEC 9126《软件工程 产品质量》。GB/T 16260—2006《软件工程 产品质量》总共分为 4 部分，分别是：GB/T 16260.1《软件工程 产品质量 第 1 部分：质量模型》、GB/T 16260.2《软件工程 产品质量 第 2 部分：外部度量》、GB/T 16260.3《软件工程 产品质量 第 3 部分：内部度量》、GB/T 16260.4《软件工程 产品质量 第 4 部分：使用质量度量》。

GB/T 16260—2006《软件工程 产品质量》标准描述了关于软件产品质量的两部分模型：一是内部质量和外部质量模型；二是使用质量模型。其中，GB/T 16260.1 规定了两种质量模型，其中外部质量模型共 6 个特性 27 个子特性，内部质量模型共 6 个特性 27 个子特性，使用质量模型共 4 个特性。GB/T 16260.2 给出了软件外部质量的度量方法（112 个度量元），GB/T 16260.3 给出了软件内部质量的度量方法（70 个度量元），GB/T 16260.4 给出了软件使用质量的度量方法（15 个度量元）。有关内部质量、外部质量和使用质量的定义请参阅本篇术语部分，下面主要介绍一下标准中所阐释的质量框架，有关具体质量属性及其度量的内容读者如果感兴趣可以参考标准的相关内容。质量模型框架可以从质量途径、产品质量和生存周期、需测量的项以及质量模型的使用这几个方面加以说明。

（1）质量途径。

软件产品质量需求一般包括对于内部质量、外部质量和使用质量的评估准则，以满足开发者、维护者、需方以及最终用户的需要。为满足软件质量需求而进行的软件产品评价是软件开发生存周期中的一个过程。软件产品质量可以通过测量内部属性，也可以通过测量外部属性，或者通过测量使用质量的属性来评价，目标就是使产品在指定的周境下具有所需的效用。

过程质量、内部质量、外部质量和使用质量，它们之间具有紧密的联系。过程质量有

助于提高产品质量,而产品质量又有助于提高使用质量。因此,评估和改进一个过程是提高产品质量的一个手段,而评价和改进产品质量则是提高使用质量的方法之一。同样,评价使用质量可以为改进产品提供反馈,而评价产品则可以为改进过程提供反馈。合适的软件内部属性是获得所需外部行为的先决条件,而适当的外部行为则是获得使用质量的先决条件(见图 2-12)。

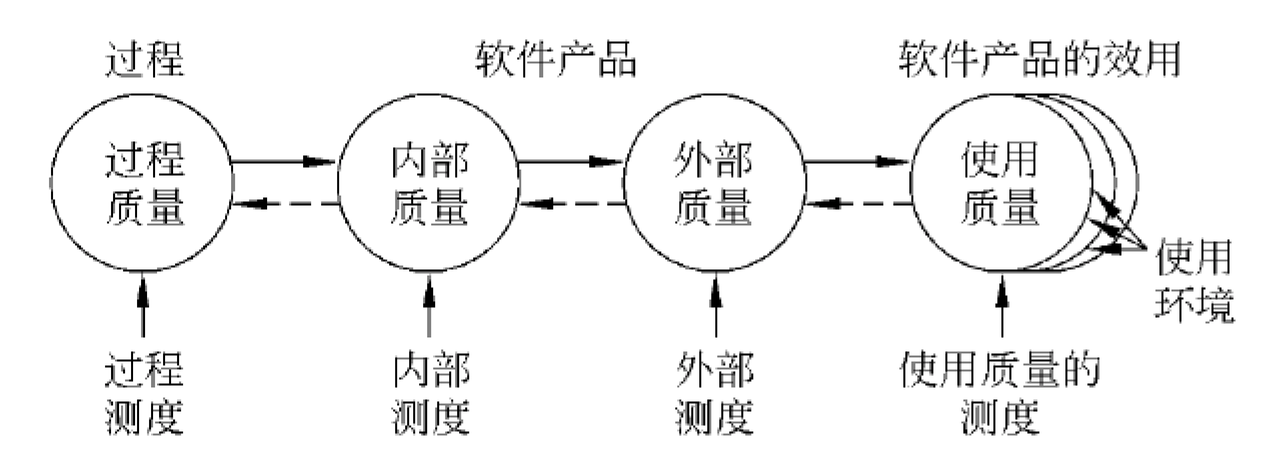


图 2-12　生存周期的质量

(2) 产品质量和生存周期。

内部质量、外部质量和使用质量的观点在软件生存周期中是变化的。例如,在生存周期开始阶段作为质量需求而规定的质量大多数是从外部和用户的角度出发的,它与如设计质量这样的中间产品质量不同,后者大多是从内部和开发者的角度来看问题的。为获得必要的质量级别,使用诸如质量的规格说明和评价这样的技术需要支持这些不同的观点。为了在生存周期的每个阶段正确地管理质量,需要对质量定义这些观点和相关技术。

在软件生存周期的不同阶段存在着关于产品质量和相关度量的不同观点,用户质量需求可通过使用质量的度量、外部度量,有时是内部度量来确定为质量需求。外部质量需求从外部视角来规定要求的质量级别。内部质量需求从产品的内部视角来规定要求的质量级别(如图 2-13)。

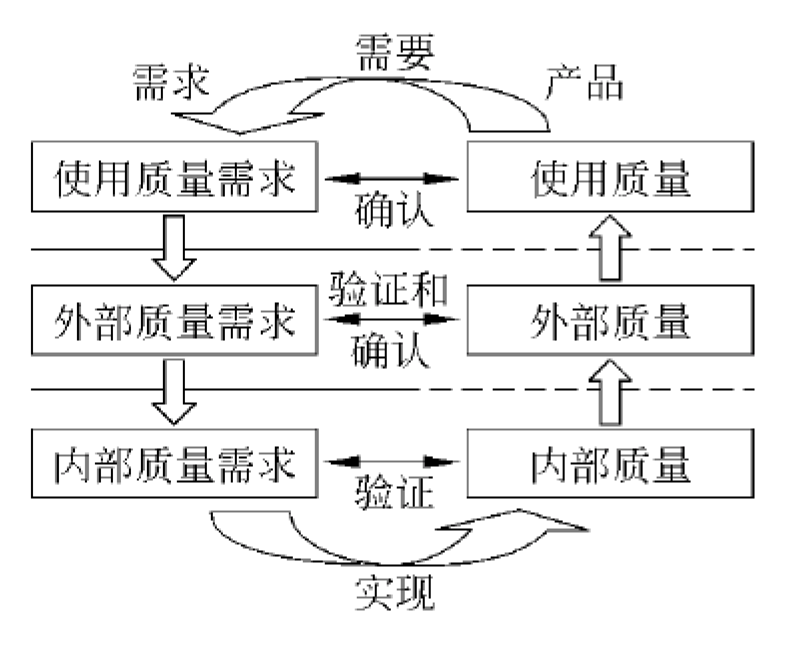


图 2-13　软件生存周期中的质量需求与质量度量

(3) 需评价的项。

项可以通过直接测量来评价,或者通过测量它们的结果来间接进行评价。例如,一个过程可以通过测量和评价它的产品来间接地进行评估,而产品可以通过测量用户的任务性能来间接地评价(采用使用质量的度量)。

(4) 质量模型的使用。

软件产品质量宜使用已定义的质量模型来评价。质量模型宜在为软件产品和中间产品设置质量目标时使用。软件产品质量应该按层次分解为一个由特性和子特性所组成的质量模型,该模型可作为与质量相关的问题清单来使用。

对大型软件产品的所有部分,测量其所有内部和外部子特性实际上是不可能的。同样,为所有可能的“用户任务”方案测量使用质量通常也是不切实际的。评价资料需要基于业务目标和产品与设计过程的性质在不同类别的测量间进行分配。

2) GB/T 18905—2002《软件工程 产品评价》

GB/T 18905 旨在提供给软件的开发者、软件的需方和独立的评价者使用,特别是供

那些负责软件产品评价的人员使用。应用 GB/T 18905 所产生的评价结果可供管理者、开发者或维护者测量软件是否符合需求,并在必要的地方予以改进。分析人员可用评价结果来建立内部度量与外部度量间的关系,过程改进人员可用评价结果来确定如何通过研究和检查项目的产品质量信息来改进过程。

GB/T 18905—2002《软件工程 产品评价》的内容等同采用自 ISO/IEC 14598《软件工程 产品评价》,说明了软件产品的评价过程,提供了评价需求和指南,为软件产品质量的测量、评估、评价提供了方法。本系列标准共分为 6 部分:GB/T 18905.1《软件工程 产品评价 第 1 部分:概述》、GB/T 18905.2《软件工程 产品评价 第 2 部分:策划和管理》、GB/T 18905.3《软件工程 产品评价 第 3 部分:开发者用的过程》、GB/T 18905.4《软件工程 产品评价 第 4 部分:需方用的过程》、GB/T 18905.5《软件工程 产品评价 第 5 部分:评价者用的过程》、GB/T 18905.6《软件工程 产品评价 第 6 部分:评价模块的文档编制》。

其中,GB/T 18905.1 概述了软件评价过程,提供评价需求和指南;GB/T 18905.2、GB/T 18905.6 属于评价支持标准,用于公司或部门级的评价管理和支持;GB/T 18905.3、GB/T 18905.4、GB/T 18905.5 给出了项目级的评价需求和指南(见图 2-14)。

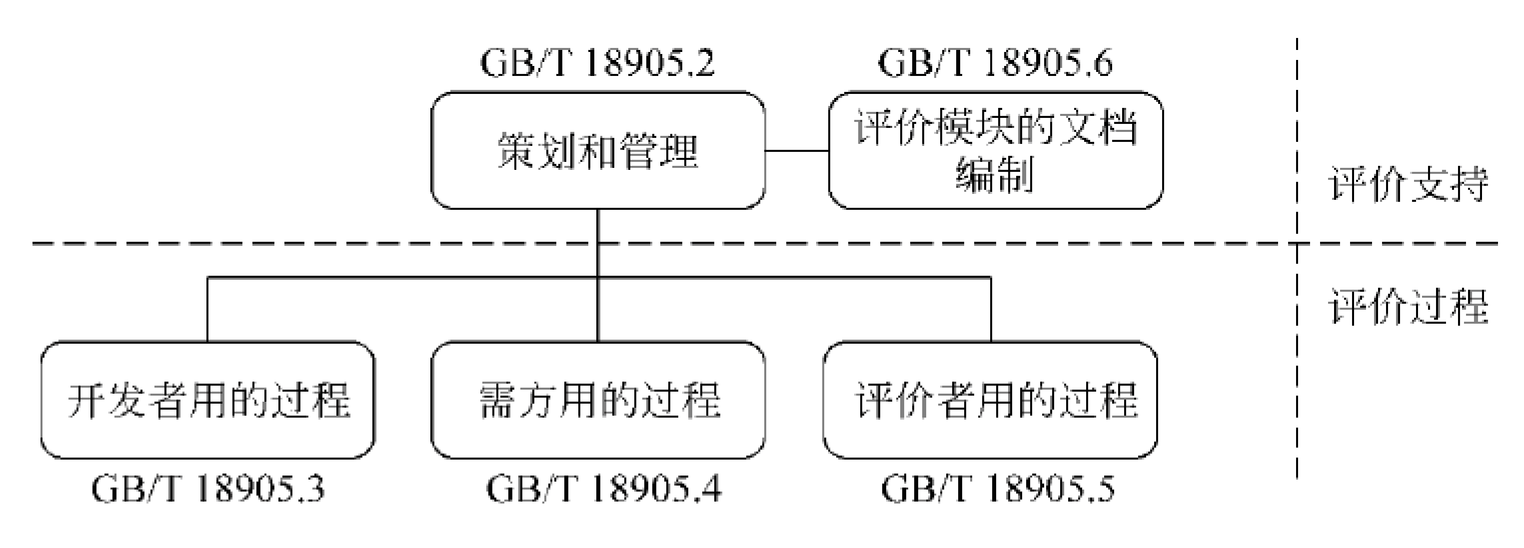


图 2-14 评价支持与评价过程

(1) 评价过程标准。

GB/T 18905 给出了 3 种不同情况下评价过程的需求和指南,它们是开发、获取和独立评价,分述如下:

① 开发者用的过程。计划开发新产品或增强现有产品,以及打算利用他们自己的技术人员进行产品评价的组织宜使用 GB/T 18905.3。这部分主要强调使用那些能预测最终产品质量的指标,这些指标将通过度量在生存期期间开发的中间产品来得到。

② 需方用的过程。计划获取或复用某个已有的软件产品或预先开发的软件产品的组织宜使用 GB/T 18905.4。该部分可用来决定接受产品或从众多可选产品中选择某个产品(产品可以是自包含的,或是系统的一部分,或者是较大产品的一部分)。

③ 评价者用的过程。对软件产品执行独立评估的评价者宜使用 GB/T 18905.5。这种评价应开发者、需方或其他方的请求来进行的。这部分将由独立执行评价的人员使用,他们通常为第三方组织进行工作。

(2) 评价支持标准。

评价支持标准为评价过程提供支持,上述每个评价过程的标准都能与 GB/T 18905.2(策划和管理)和 GB/T 18905.5(评价模块的文档编制)结合起来使用。

① 策划和管理。GB/T 18905.2 包含对软件产品评价的支持功能的需求和指南。这

种支持与策划和管理软件评价过程及相关的活动有关，包括组织内评价专业知识的开发、获取、标准化、控制、转换和反馈。

② 评价模块。GB/T 18905.6 为编制评价模块的文档提供指南。这些模块包括质量模型的规范(即特性、子特性和相应的内部或外部度量)、与模型计划的应用有关的数据和信息、与模型的实际应用有关的信息。每种评价均应选择适当的评价模块，某些情况下，可开发新的评价模块。

(3) 评价过程。

要评价软件的质量，首先要确立评价需求，然后规定、设计和执行评价(见图 2-15)。

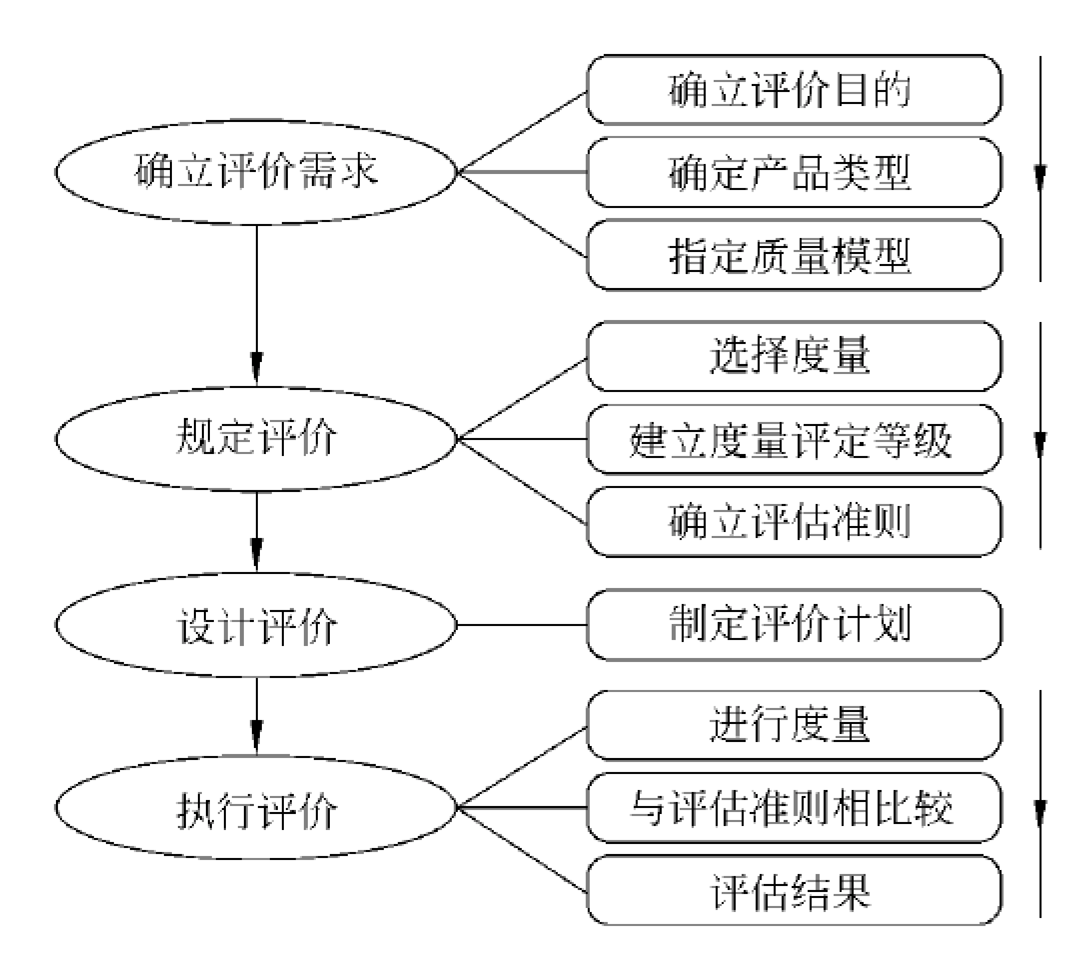


图 2-15 评价过程模型

3) 度量与评价标准的发展和 SQuaRE

前面分别介绍了有关软件度量和评价的两个标准 GB/T 16260—2006《软件工程 产品质量》和 GB/T 18905—2002《软件工程 产品评价》，事实上在软件测试领域还有一个常用的国家标准 GB/T 17544—1998《信息技术 软件包 质量要求和测试》，这个标准主要介绍了软件包的质量要求和测试细则。但是因为这个标准发布年份距今时间已经比较久远，即将面临改版，因此在本书中不再予以细述。实际上，即便是 GB/T 16260—2006《软件工程 产品质量》和 GB/T 18905—2002《软件工程 产品评价》这两个较新的标准在实际应用中仍然存在着一些不足，部分遗留问题仍然等待解决。例如：

- 需要独立的架构；
- 需要独立的系列名称和代号；
- 对于度量标准的使用没有指导；
- 缺少度量的基础标准；
- 没有质量需求标准。

为了解决上述问题，ISO 正在重新研究和组织这些标准的内容与结构，并有望在未来几年予以发布。这就是 Software Engineering-Software Product Quality Requirements and Evaluation，简称 SQuaRE，代表了度量与评价标准的最新发展趋势。

根据目前所获得的信息，可以知道 SQuaRE 至少包含了以下要素：

- 引入了新的 SQuaRE 标准的体系结构（见图 2-16）；
- 引入了新的通用参考模型（见图 2-17）；
- 引入了各个部分专门的详细指南；
- 引入了质量度量部分内部测量要素；
- 引入了质量需求的标准；
- 合并和修订了评价过程；
- 引入了构成示例中的实际使用指南。

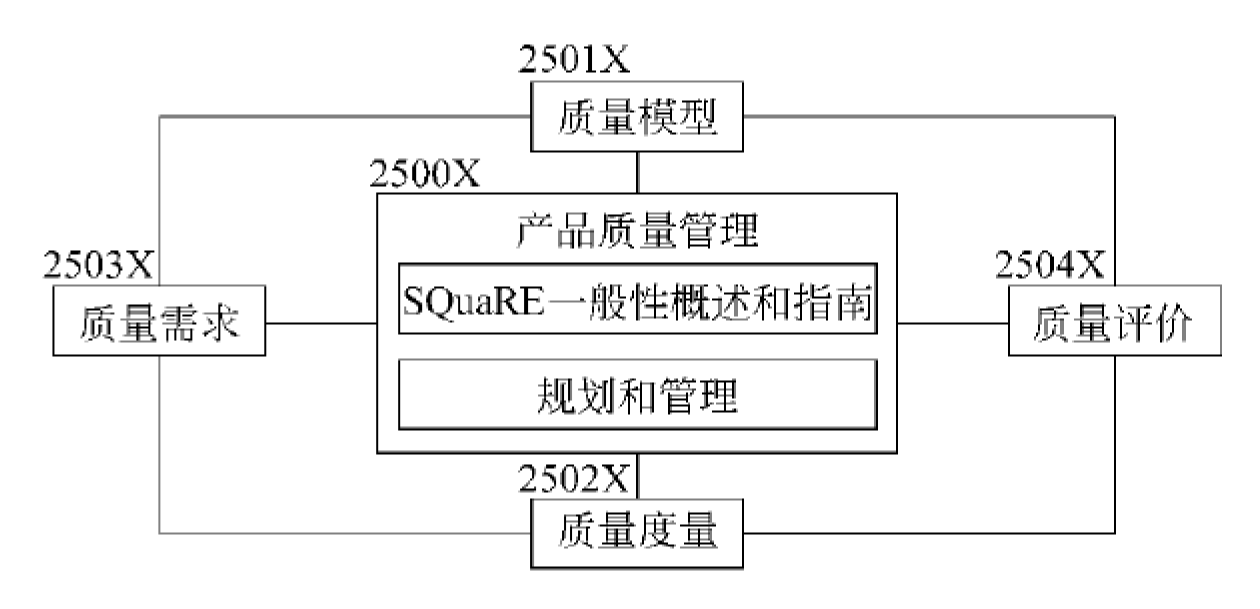


图 2-16　SQuaRE 框架

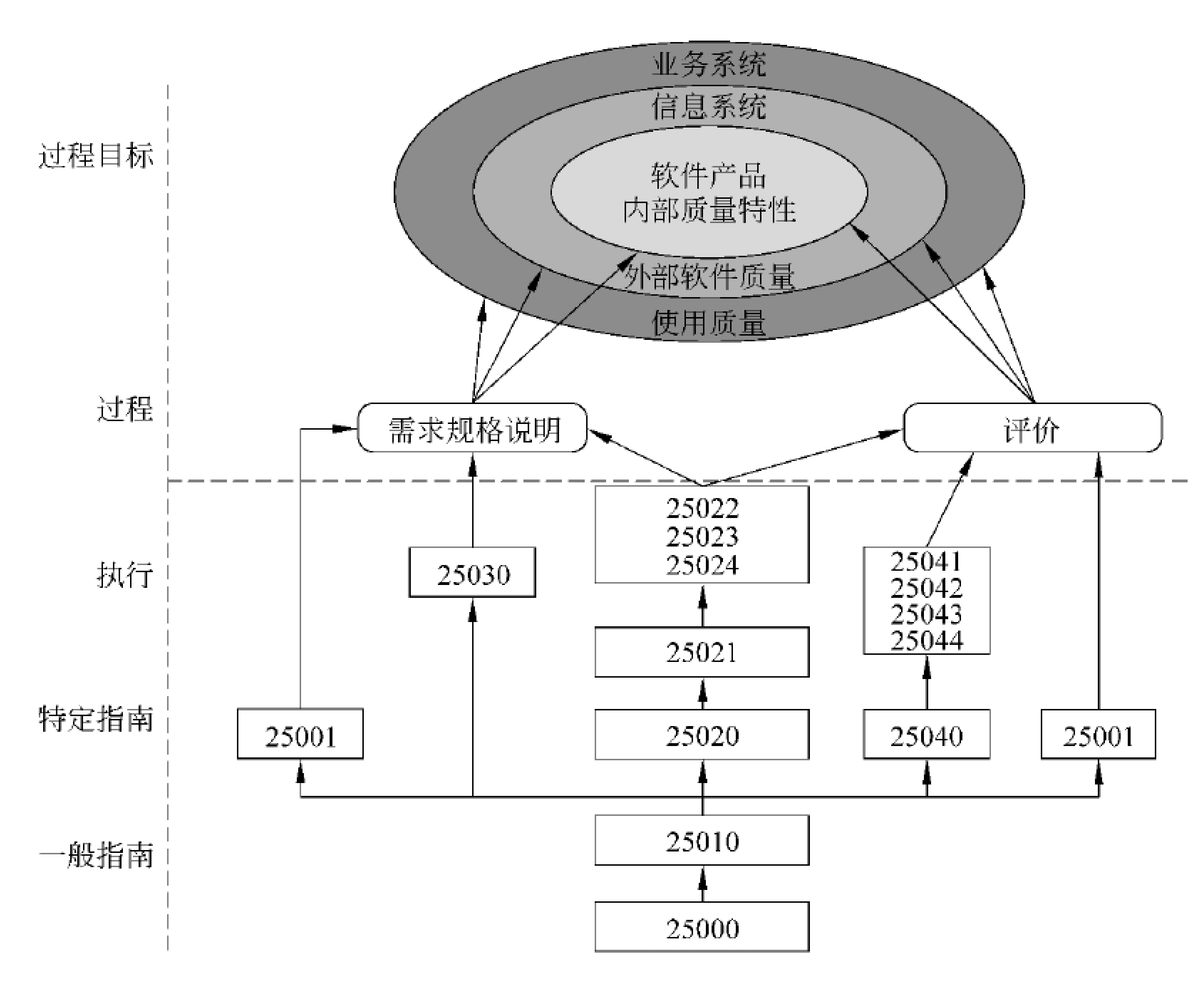


图 2-17　SQuaRE 通用参考模型

在 SQuaRE 中，ISO/IEC 9126《软件工程 产品质量》（对应 GB/T 16260—2006《软件工程 产品质量》）和 ISO/IEC 14598《软件工程 产品评价》（对应 GB/T 18905—2002《软件

工程 产品评价》)的内容被分散到 250××系列标准中，例如 ISO/IEC 9126《软件工程 产品质量》质量模型部分被分到 25000：SQuaRE 指南和 25010：质量模型与指南中讲述。另外，前面所说的 GB/T 17544—1998《信息技术 软件包 质量要求和测试》在 SQuaRE 中将对应 25051 标准。

本章小结

软件工程经过几十年的发展，目前已经进入了一个新的阶段。新的软件工程方法、技术、过程、工具层出不穷，但始终围绕方法、过程、工具这三个软件工程要素进行研究和实践。因此，把握和理解软件工程的三要素和基本原理，是有效实践软件工程的重要条件。

软件工程有关标准为软件过程的度量改进和软件产品的质量评价提供了根本性的理论和方法，为软件测试的发展指明了道路，也提供了发展基础。

CHAPTER

第 3 章 软件过程能力评估与 CMM/CMMI

软件过程能力是衡量软件开发组织对软件质量的保障能力的重要标志。CMM/CMMI 是 1987 年由美国卡内基梅隆大学软件工程研究所提出的，为软件过程能力评估提供了有效的框架，并得到了最广泛的应用，成为业界事实上的标准。本章对 CMM、CMMI 进行简要介绍，以使读者对这一国际上流行的软件过程评估模型有一个初步的了解，并从 CMM/CMMI 这个高度对软件质量管理、软件测试等的关系有一个深刻的认识。

3.1 CMM/CMMI 综述

如前所述 20 世纪 60 年代提出的软件危机问题，到了 20 世纪 80 年代仍然没有真正缓解，因为要求软件解决的问题的复杂性增长之快，远远超过了人们开发及维护软件的能力。随着这些问题的不断加剧，软件开发组织逐渐意识到应该在软件的管理和开发过程这一源头来采取改进措施，以达到控制和提高软件的质量的目的。与此同时，软件用户(如政府机构、金融机构)也希望参照一定的标准来评估软件开发商的能力。然而，这种改进工作不可能一蹴而就，需要持续不断地进行。软件过程改进主要是在一系列微小的、不断发展的步骤中实现的，并建立在企业文化变化的基础上。

软件过程改进本身是一种非常复杂的过程，不可能被简单地、轻易地表示和确定。为了正确和有序地进行软件过程中的活动，必须为软件过程建立一种能够良好地描述和表示的模型。依据这种模型，即可容易地确定各阶段需完成的任务、实现任务的评估方法以及表达各阶段间的次序和关系。

1986 年 11 月，美国卡内基梅隆大学的软件工程研究所(Software Engineering Institute，SEI)开始设计软件过程成熟度框架，以帮助软件组织改进其软件过程，同时也可作为软件过程能力评估的模型，SEI 最终于 1987 年正式推出 SW-CMM 框架，并于 1991 年发布 SW-CMM1.0 版本，也就是我们所说的 CMM(Capability Maturity Model，软件过程能力成熟度模型)，目前使用的 SW-CMM 的版本是 1.1，是 SEI 于 1993 年发布的。

CMM 问世以来，一方面在软件过程改进方面为用户提供了有效的解决方案；另一方面也出现了许多难以应付的现象，这是因为随着软件的应用领域的逐渐扩大和复杂度的日趋增强，使得 CMM 仅关注传统的软件工程范畴的问题这一特点变得不再适应时代的发展。同时，CMM 也需要同其他同类标准（如 ISO 相关标准）进行协调。因此，扩大 CMM 的适用领域，成为 SEI 对 CMM 所进行的完善工作的重要内容。从 1998 年之后，SEI 应美国国防部的要求转向 CMMI 的研发工作。

CMMI(Capability Maturity Model Integration)即能力成熟度集成模型，是美国国防部的一个设想，它们想把现在所有的以及将被发展出来的各种能力成熟度模型，集成到一个框架中去。这个框架有两个功能，第一，软件采购方法的改革；第二，建立一种从集成产品与过程发展的角度出发、包含健全的系统开发原则的过程改进。就软件而言，CMMI 是 SW-CMM 的修订本，于 2000 年首次发布，目前广泛使用的是 CMMI1.2，这个版本是在 2006 年 8 月发布的。CMMI 包括以下内容。

- 软件 CMM(Capability Maturity Model for Software，SW-CMM)：软件工程的对象是软件系统的开发活动，要求实现软件开发、运行、维护活动系统化、制度化、量化。
- 系统工程 CMM(System Engineering Capability Maturity Model，SE-CMM)：系统工程的对象是全套系统的开发活动，可能包括也可能不包括软件。系统工程的核心是将客户的需求、期望和约束条件转化为产品解决方案，并对解决方案的实现提供全程的支持。
- 集成产品开发 CMM(Integrated Product Development Capability Maturity Model，IPD-CMM)：集成的产品和过程开发是指在产品生命周期中，通过所有相关人员的通力合作，采用系统化的进程来更好地满足客户的需求、期望和要求。
- 采购 CMM(Supplier Sourcing Capability Maturity Model ，SS-CMM)：采购的内容适用于那些供应商的行为对项目的成功与否起到关键作用的项目。主要内容包括：识别并评价产品的潜在来源、确定需要采购的产品的目标供应商、监控并分析供应商的实施过程、评价供应商提供的工作产品以及对供应协议和供应关系进行适当的调整。

除了以上模型之外，SEI 还发布了人员 CMM(People Capability Maturity Model，P-CMM)、群组软件过程(Team Software Process)、个体软件过程(Personal Software Process)等模型，作为 CMMI 的补充和扩展。

CMM/CMMI 从颁布至今，在以下三个领域得到了广泛认可和应用。

① 软件过程评估(Software Process Assessment，SPA)：指出该企业所面临的与软件过程有关的、最急需解决的问题，以便改进。

② 软件过程改进(Software Process Improvement，SPI)：帮助软件企业对其软件过程向更好的方向改变。

③ 软件能力评价(Software Capability Evalutions，SCE)：鉴别软件承包者的能力资格；或检查/监督正用于软件制作的软件过程的情况。

3.2 CMM/CMMI 基本框架

3.2.1 CMM

CMM 模型将软件过程能力成熟度划分为五个级别（如图 3-1 所示），每个级别的软件过程的特征如下：

① 初始级（initial level）。软件过程杂乱无章，有时甚至是混乱的。几乎没有明确的过程定义，成功完全依赖于个人努力和英雄人物。

② 可重复级（repeatable level）。建立了基本项目管理过程来跟踪成本、进度和机能。有必要的过程准则来重复以前同类项目的成功。

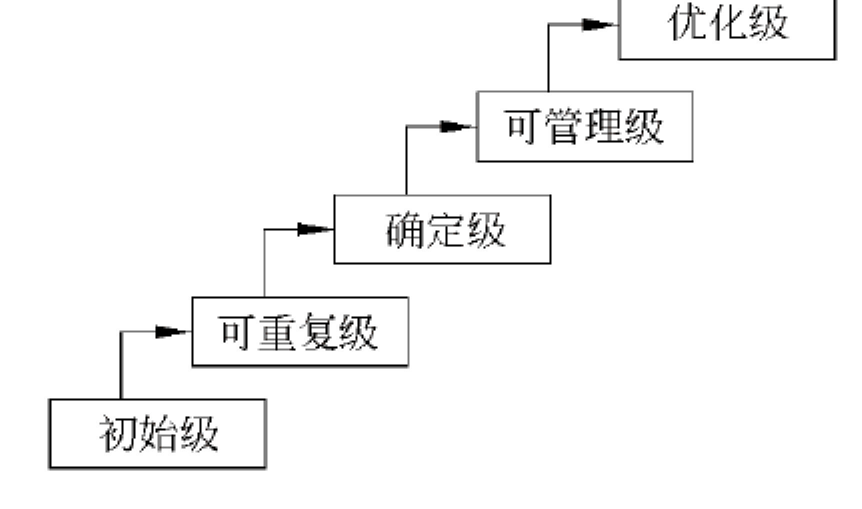


图 3-1　CMM 的五个成熟度等级

③ 确定级（defined level）。管理过程和软件过程都文档化、标准化，综合成整个软件开发组织的标准软件过程。所有项目都采用裁剪后的标准软件过程来进行软件开发和维护。

④ 管理级（managed level）。制定了软件过程和产品质量的详细的度量标准。开发组织的成员理解和控制软件过程和产品的质量。

⑤ 优化级（optimizing level）。通过过程质量的反馈和来自新观念、新科技的反馈，加强量化分析，过程不断持续地改进。

SEI 通过能力成熟度等级的划分，来衡量软件开发组织的软件过程可信度，用以回答美国联邦政府要求提供评价软件开发商能力的要求。CMM 中的每一个等级，标示了软件组织的过程能力，每种级别提供更好成熟度的软件过程改进方向。按照 CMM 的要求，可以系统地实现持续的软件过程改进，使软件组织的软件能力持续地得到提高。

3.2.2 CMMI

为了兼顾软件开发组织对于某一个软件过程领域的改进活动，CMMI 在吸取了 SW-CMM 和美国电子行业协会临时标准（EIA/IS）731 的表示方法的基础上，将软件过程能力成熟度分为阶段式和连续式这两种在逻辑上等价的表示形式（如图 3-2 所示）。SW-CMM 软件能力成熟模型是阶段式的模型，SE-CMM 系统工程模型是连续式模型，而 IPD-CMM 集成产品开发模型结合了阶段式和连续式两者的特点。

阶段式表示法主要用于对软件组织的过程能力成熟度进行评估，基本沿袭 SW-CMM 模型框架，仍然保持五个“成熟度等级”，但过程域（PA）做了一些调整和扩充。阶段式表示法中的每个“成熟度等级”都被分解为过程域、特殊目标和特殊实践、通用目标和通用实践、共同特性——每个等级都有几个过程域组成，这几个过程域共同形成一种软件过程能力。每个过程域，都有一些特殊目标和通用目标，通过相应的特殊实践和通用实践来实现这些目标。当一个过程域的所有特殊实践和通用实践都按要求得到实施，就能实

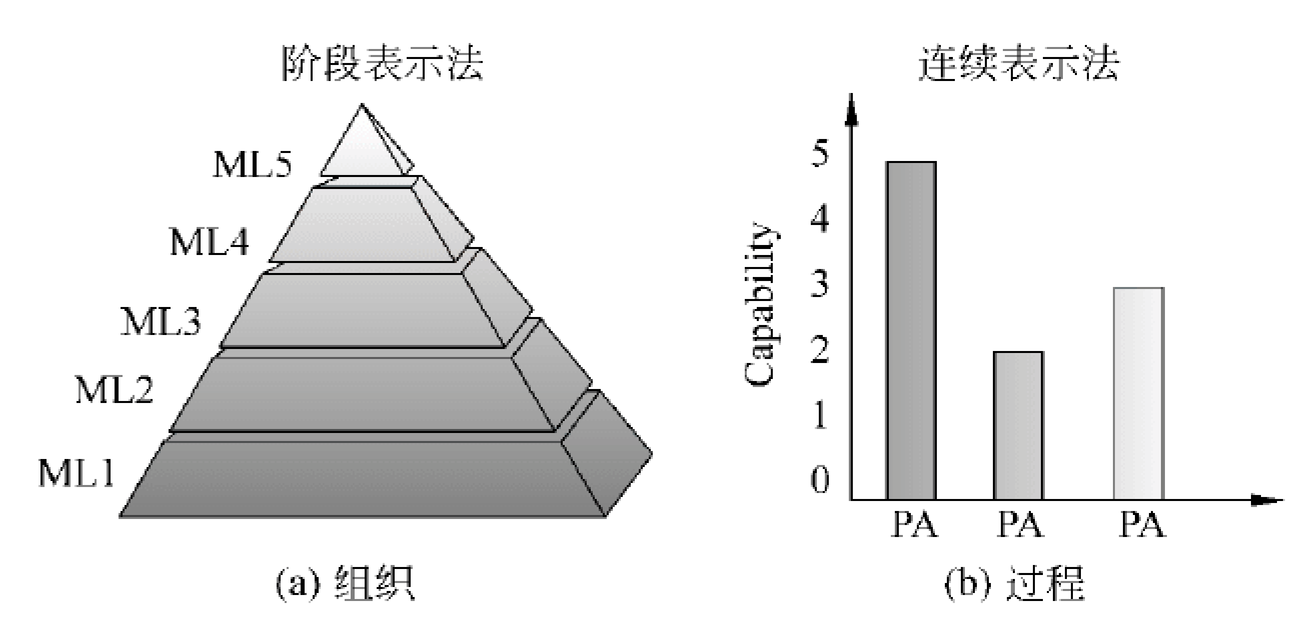


图 3-2 CMMI 的阶段式和连续式表示法

现该过程域的目标。

CMMI 阶段式各等级过程域如表 3-1 所示。

表 3-1 CMMI 阶段式各等级过程域

成熟度等级	过程域(PA)
L1 初始级	无
L2 可重复级	需求管理、项目计划、配置管理、项目监督和控制、供应商合同管理、度量和分析、过程和产品质量保证
L3 已定义级	需求开发、技术解决方案、产品集成、验证、确认、组织级过程焦点、组织级过程定义、组织级培训、集成化项目管理、风险管理、集成化的团队、决策分析和解决方案、组织级集成环境
L4 已管理级	组织级过程性能、项目定量管理
L5 优化级	组织级改革和实施、因果分析和解决方案

连续式表示法将软件过程领域分为过程管理、项目管理、工程、支持，主要用于软件组织的某一个过程领域的能力度的评估。连续式模型共有六个能力度等级，分别是：

- 0 级——不完整级；
- 1 级——执行级；
- 2 级——管理级；
- 3 级——定义级；
- 4 级——量化管理级；
- 5 级——最佳化级。

每个能力度等级对应到一个一般目标，以及一组一般执行方法和特定方法。

两种表示法的差异反应了为每个能力和成熟度等级描述过程而使用的方法，他们虽然描述的机制可能不同，但是两种表示方法通过采用公用的目标和方法作为需要的和期望的模型元素，而达到了相同的改善目的。两种表现方式(连续的和阶段的)从他们所涵盖的过程区域上来说并没有不同，不同的是过程区域的组织方式以及对成熟度(能力)级别的判断方式。

3.3 CMM/CMMI 与软件测试

CMM/CMMI 的目标是为提高组织过程和管理产品开发、发布和维护的能力提供保障,帮助组织客观地评价自身能力成熟度和过程域能力,为过程改进建立优先级以及执行过程改进,归根结底也是为了提高软件产品的质量。CMM/CMMI 的质量管理理念是“产品的质量在很大程度上取决于用以开发和维护该产品的过程的质量”。因此,CMM/CMMI 推行基于过程的软件质量管理,质量管理活动是 CMM/CMMI 实施过程中的核心内容,无论是质量保证、度量与分析、验证、确认、质量管理等等过程域,无一不贯穿于整个 CMM/CMMI 的实施全过程。

通过实施 CMM/CMMI,有助于改进软件产品的质量、改进项目满足预定目标的能力、减少开发成本和周期、降低项目风险、提高组织过程能力,从而提高竞争力、市场占有率、效益。实施不同等级的 CMM/CMMI,对于按照每功能点来计算的软件缺陷率的降低具有显著的作用(如表 3-2 所示)。

表 3-2 实施 CMM 对软件质量的影响

CMM 等级	隐含的缺陷(每功能点)	缺陷消除率	交付的缺陷
1	5.00	85%	0.75
2	4.00	89%	0.44
3	3.00	91%	0.27
4	2.00	93%	0.14
5	1.00	95%	0.05

数据来自参考文献[6]。

CMM/CMMI 基于过程的软件质量管理主要包括质量保证和质量控制两大方面。软件质量保证是由(相对)独立的质量管理人员在项目的整个开发周期中对项目所执行的过程和产生的工作产品进行监督和检查,确保其符合预定的要求。软件质量保证的目的是确保过程得到有效的执行,并推进过程改进,并就项目过程的执行情况和所构造的产品向管理者提供适当的可视性。软件质量控制是指为评价和验证已开发的产品而执行的活动和技术,包括验证产品是否满足质量要素的要求,以及产品(包括生命周期的工作产品)是否具有可接受的质量。软件质量控制所采取的主要技术便是软件测试,通过软件测试,来验证产品是否符合技术文档预期的特性、功能和性能等要求,并识别产品的缺陷。

虽然目前在 CMM/CMMI 中没有明确软件测试作为一个独立的过程域,但是在 CMM/CMMI 中很多地方都涉及了软件测试内容。例如,在 CMMI 的“确认”和“验证”这两个过程域中 CMMI 列出了以下实践。

验证特定目标及实践摘要

SG 1 确认准备

SP 1.1 选择需确认之产品

SP 1.2 建立确认环境
SP 1.3 建立确认程序与准则
SG 2 确认产品或产品组件
SP 2.1 执行确认
SP 2.2 分析确认结果

验证特定目标及实践摘要

SG 1 验证准备
SP 1.1 选择需验证之工作产品
SP 1.2 建立验证环境
SP 1.3 建立验证程序及准则
SG 2 执行同行审查
SP 2.1 准备同行审查
SP 2.2 进行同行审查
SP 2.3 分析同行审查资料
SG 3 验证工作产品
SP 3.1 执行验证
SP 3.2 分析验证结果

如果对软件测试有些了解，就会发现这些要求似曾相识，事实上在从事软件测试时，这些工作都已经包含在了软件测试生命周期中，每个事项在软件测试生命周期中都可以对号入座。

验证特定目标及实践摘要

SG 1 确认准备
SP 1.1 选择需确认之产品
SP 1.2 建立确认环境
SP 1.3 建立确认程序与准则

> **测试计划：**
> 测试范围识别、测试准备、工作流程、测试准入与完成准则

SG 2 确认产品或产品组件
SP 2.1 执行确认
SP 2.2 分析确认结果

> **测试执行：**缺陷记录、缺陷分析、缺陷修改与回归
> **测试报告**

验证特定目标及实践摘要

SG 1 验证准备
SP 1.1 选择需验证之工作产品
SP 1.2 建立验证环境
SP 1.3 建立验证程序及准则

> **测试计划：**
> 测试范围识别、测试准备、工作流程、测试准入与完成准则

SG 2 执行同行审查

SP 2.1 准备同行审查
SP 2.2 进行同行审查
SP 2.3 分析同行审查资料
SG 3 验证工作产品
SP 3.1 执行验证
SP 3.2 分析验证结果

测试执行：缺陷记录、缺陷分析、缺陷修改与回归
测试报告

本章小结

当前，软件工程研究者和实践者对于CMM/CMMI的热衷和追捧，仍然是以软件质量的提高和软件开发组织能力的改进为直接驱动力的。伴随着CMM/CMMI的在软件企业的大规模实施和等级评估，软件质量得到了迅速强化，软件测试作为检测软件质量所倚重的手段在期间也获得了广泛认可，这一切为软件测试的长足发展奠定了坚实基础，也营造了优良的发展环境。这也正是本书拿出这一章的篇幅对CMM/CMMI进行介绍的一个重要原因，希望大家在学习软件测试的时候能够以更广阔的视角去了解和认识软件测试，知其然也知其所以然。

PART 2

第二篇

软件测试概述

第一篇中简单讲述了软件的发展及其工程化的演变，扼要介绍了在软件工程中有着重要应用的几个标准，对当今使用比较广泛的软件过程能力评估模型 CMM/CMMI 做了简单说明。作为一个初涉软件测试领域的准测试人员来说，这是一个基本的开始，为什么这么说呢？古人云，“不识庐山真面目，只缘身在此山中”，用在软件测试中就是说软件测试的工作和关联性技术非常广泛，一旦从事软件测试，往往会陷入各种技术的学习和积累、产品问题的查找中，但是几年下来问一问他什么是软件测试时，却常常并不能解释清楚。这无疑是一件非常令人感到沮丧的事情，从局部到整体的困难正向我们展露着软件测试的尴尬，本篇将阐释软件测试在软件工程中的角色，这正是它或生或灭的来时之路。

那么接下来，将在这个基础上开始软件测试的入门介绍，回答“什么是软件测试”这样一个基本问题。同时，本篇力争全面地介绍软件测试的各种分类方法、各类过程模型，给读者一个全面权威的认知，避免在将来的深入学习过程中产生过多的概念困扰。

本篇名词解释

软件质量(quality of software)：软件质量是与软件产品满足明确或隐含需求的能力有关的特征和特性的总和。

质量控制(quality control)：为达到质量要求所采取的作业技术和活动称为质量控制。这就是说，质量控制是为了通过监视质量形成过程，消除质量环上所有阶段引起不合格或不满意效果的因素。

软件测试过程模型(**model for software testing process**):软件测试的工作框架,用于指导软件测试过程,在软件测试过程中使用合理的测试模型可以降低成本,提高效率。

回归测试(**regression testing**):为了确保软件在完善后、缺陷修复后或者任何其他变更不会导致原有功能的失效,通常需要重新对软件发生改变的部分进行测试,这个过程称为回归测试。

测试用例(**test case**):简单来说就是预先编制的一组系统操作步骤和输入数据、执行条件以及预期结果,用以验证某个程序是否满足某个特定需求的文字。

CHAPTER

第4章 软件质量

随着软件产业的发展，软件质量逐步被视为企业生命。很多企业都将业务建立在软件系统上，软件与业务密切结合，高质量的软件系统有助于企业增强自身实力，在市场竞争中赢得优势。因此，正确理解并有效提高软件质量，成为至关重要的目标。

4.1 什么是软件质量

在介绍“软件质量”之前，首先了解一下“质量”的概念。通常将“质量”等同于属性，片面认为属性好就是质量好，但事实证明属性好的实体不一定能胜任并完成任务。这里，将“质量”定义为“一组固有特性满足明确或隐含要求的能力”，属性是固有的，与要求相比较的满足程度才能反映出质量的好坏。另外，质量也不仅仅指产品质量，还包括活动或过程的工作质量、制度/体系的运行质量等。

“软件质量”属于一种产品质量，反映为软件产品满足明确或隐含要求的程度，包括软件与确定的功能及性能需求的一致性、与成文的开发标准的一致性、与所有专业开发的软件所期望的隐含特性的一致性等。

“软件质量”既然体现为产品满足要求的程度，那不同的人从不同的角度就会有不同的理解。例如，同一软件产品，对用户来说，“软件质量”体现为软件产品是否满足用户自身的需求；对开发者来说，“软件质量”体现为软件产品是否与需求说明一致；对产品本身来说，“软件质量”体现为软件产品的内在特性；从价值的角度上看，“软件质量”体现为客户是否愿意购买该软件产品。GB/T 16260—2006《软件工程 产品质量》(idt：ISO/IEC 9126)中将软件质量分为内部质量、外部质量、使用质量，分别从内部视角、外部视角、用户观点来衡量软件质量。

软件质量是各种特性的复杂组合，软件质量特性包括正确性、精确性、健壮性、可靠性、容错性、效率、易用性、安全性、可扩展性、可复用性、兼容性、可移植性、可测试性、可维护性、灵活性等，软件质量能够随着应用或用户提出质量要求的不同而不同。

4.2 软件质量管理

软件质量正逐步被软件企业视为其生命，软件质量管理开始在软件组织内全面开展，管理模式也从单纯的质量检验发展到全面质量管理、六西格玛质量管理和零缺陷管理等新的管理理论、方法和体系。

4.2.1 质量管理基础

这里首先介绍质量管理的基础知识。质量管理是确定质量方针、目标和职责并在质量体系中通过质量策划、质量保证、质量控制和质量改进使其实施管理职能的全部活动。

1. 质量策划

质量策划是为确定质量和质量体系要素应用的目标和要求的活动，包括产品策划、管理和作业策划、编制质量计划并为质量改进作准备等过程。

2. 质量保证

质量保证是为了提供足够的信任，以表明实体能够满足质量要求，而在质量体系中实施并根据需要进行证实的全部有计划的和系统的活动。

3. 质量控制

质量控制适用于对任何有关质量活动的控制，对阶段性成果进行测试、验证，为质量保证提供参考依据。

4. 质量改进

质量改进是为向本组织及其顾客提供增值效益，在整个组织范围内所采取的提高活动及过程的效益和效率的各种措施。

4.2.2 软件质量管理的手段和方法

软件质量管理要求保证软件项目能够兑现其关于满足各种需求的承诺，管理对象包括过程质量、软件产品质量，管理过程主要包括制定软件质量计划、软件质量保证、软件质量控制 3 个过程域。

1. 制定软件质量计划

软件质量计划是整个项目质量策划的结果之一，用于确定软件项目应该达到的质量标准和为达到这些质量标准对应的计划、安排和方法。

编制软件质量计划通常采用流程图、因果分析图等方法对软件项目进行分析，确定项目中监控的关键元素，设置合理的见证点、停止待检点，并制定质量标准。软件质量计划的主要内容包括：

- 软件质量计划目的、术语、参考资料等；

- 质量管理机构及其任务、职责;
- 整个软件过程中需产生的文档及其评审、检查准则;
- 软件开发过程中要用到的标准、条例和约定,以及监督措施;
- 评审和检查,包括技术和管理两方面,并编制或引用有关的评审和检查准则;
- 软件配置管理;
- 工具、技术和方法;
- 质量记录内容及存储;
- 必须的培训活动;
- 风险管理。

2. 软件质量保证

软件质量保证是贯穿软件项目整个生命周期的有计划和有系统的活动,经常针对整个项目质量计划的执行情况进行评估、检查与改进,向管理者、顾客或其他方提供信任,确保项目质量与计划保持一致。

确保软件项目过程遵循了对应的标准及规范要求且产生了合适的文档和精确反映项目情况的报告,其目的是通过评价项目质量建立项目达到质量要求的信心。软件质量保证活动主要包括评审项目过程、审计软件产品,就软件项目是否真正遵守已制定的计划、标准和规程等,给管理者提供可视性项目和产品可视化的管理报告。

3. 质量控制

软件质量控制是对软件项目的阶段性成果进行测试、验证,确定项目成果与相关质量标准是否相符,同时,确定消除不符的原因和方法,目的是控制产品质量、及时纠正缺陷。软件质量控制活动主要包括:

- 技术评审:在软件开发各阶段结束后,都要组织评审,对质量进行评价,尽早发现软件开发过程中可能引起软件质量问题的潜在错误。
- 代码走查:通过浏览代码,检查语法结构、调用关系,检查软件代码编写质量、是否与设计相符、是否与开发需求一致、是否符合编码规范、是否存在明显的缺陷等。
- 软件测试:软件测试是软件质量控制的重要手段,包括单元测试、集成测试、系统测试、验收测试,通过充分测试可以发现软件中大多数隐藏的缺陷。

4.3 软件质量与软件开发、测试

随着社会信息化的发展,软件的触角深入到社会生活的各个领域,几乎无处不在。软件的重要性导致人们对于软件的质量有了更高要求和更多关注,围绕软件质量的质量管理研究被空前强化。

软件产品同其他任何事物一样,也需要经历孕育、诞生、成长、成熟、衰亡等一系列过程,这个过程称为软件开发生命周期(Software Pevelopment Life Cgcle, SDLC),通常软

件生命周期包括需求分析、设计、编码、测试和维护等活动，而在各个活动中都可能产生错误，如需求定义错误、代码语法错误、缺少功能等，因此，软件测试不应局限于软件开发的某一个独立的阶段，而应贯穿于软件定义与开发的整个过程，要对过程中的每个阶段性成果进行测试，尽可能多地消除潜在的错误，以提高软件质量。从上面这段描述中，可以看出自顶向下的开发“瀑布”与自底向上的测试“喷泉”，构成了最为基本的开发与测试的关系（见图4-1），因为它们交叉形如英文字母V，故在软件测试领域，这种结构的测试过程被称为“V模型”（有关V模型的详见第6章）。

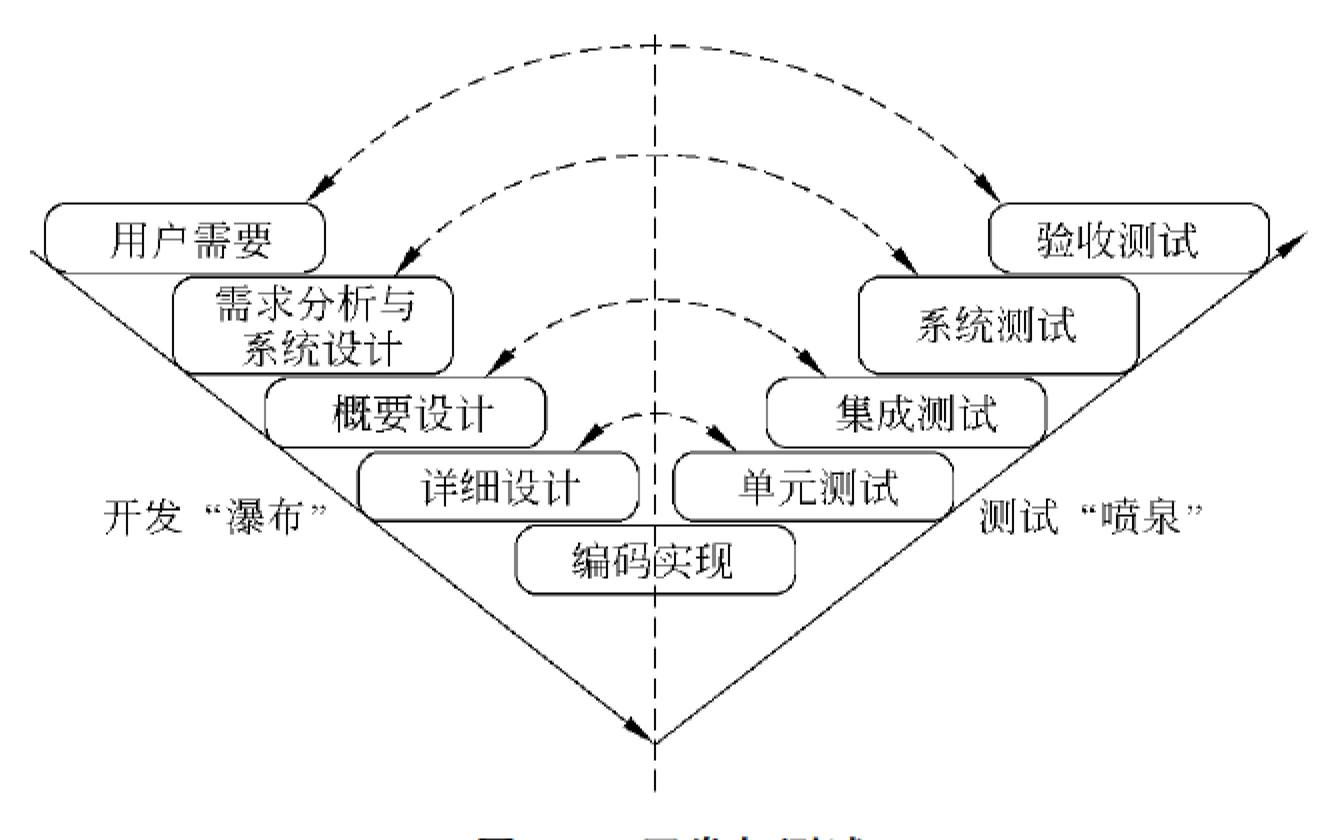


图4-1　开发与测试

在整个软件过程中，软件开发与软件测试占据了大部分的时间，无疑它们是影响软件质量的主要因素。很多时候，人们谈论开发与测试的关系时往往会简单地认为它们是个矛盾体，因为一个是要建设，而另外一个是要搞破坏，显然这种理解有很大的片面性，如果把它们统一放到软件质量这样一个层面上来看它们是统一的，不论是开发还是测试它们最终目的都是为了获得一款高质量、高满意度的软件产品。事实上，开发是影响软件质量的决定因素，合理的需求、准确的理解、科学的设计是获得好的软件质量的先决条件；软件测试是重要的质量保证手段，综合运用各类软件技术全面地对软件过程的各个阶段进行检测，通过测试数据揭示软件质量状况。

因此，从这个意义上说提高软件产品质量的途径主要有二个，一个是寻求改进软件开发过程质量的方法，以CMM/CMMI为代表的软件成熟度模型和GB/T 8566给出的软件生存周期过程为主要代表，从规范软件过程出发，持续改进软件过程，来保证软件产品开发的质量；另一个是面向软件生存周期建立全面的软件测试体系，对产品（包括阶段性中间产品）进行测试、度量和评价，用以验证所开发的软件产品是否符合规定的要求。

本章小结

本章介绍了与软件测试密切相关的基本概念及主要知识点，为读者理解软件测试奠定基础。

软件质量的重要性是不言而喻的,那么如何才能保证软件质量呢? 4.2 节介绍了软件质量管理的手段和方法,详细介绍质量计划、质量保证、质量控制等常见手段的输入、采用的技巧和手段、输出等过程。

本章最后讨论了软件质量与软件开发、软件测试的关系,通过这种分析使读者对于软件测试有一个正确的认识。

CHAPTER

第 5 章 软件测试基础

软件测试贯穿于软件项目的整个生命过程，是软件质量保证的重要手段。软件测试通过在软件项目各个阶段进行的不同目的及内容的测试活动，来保证各个阶段的正确性。对整个软件工程来说，软件测试在保证产品质量、科学控制成本、保证软件可靠性、提高企业竞争力等方面都有着十分重要的作用。

5.1 软件测试的历史及演变

软件测试是伴随着软件的产生而产生的。早期的软件规模很小、复杂程度低、开发过程混乱无序，测试的含义也比较狭窄，测试等同于“调试”，目的是纠正软件中已知的故障，测试投入很少，介入时间晚，多在产品基本完成时才进行测试。

1957 年，软件测试开始作为一种发现软件缺陷的活动，逐步区别于调试。但此时的软件测试仍存在介入时间晚的问题，同时，缺乏有效的测试方法，测试过程中主要依靠“错误推测”的方法来寻找软件缺陷。因此，这一阶段的大量软件在交付后仍存在很多问题，产品质量无法保证。

20 世纪 70 年代，软件虽然仍不复杂，但“软件工程”的概念开始频繁出现，软件测试的探索者们提出了“在软件生命周期的开始阶段就根据需求制订测试计划”的建议。1972 年，软件测试领域的先驱 Bill Hetzelt 博士在美国的北卡罗来纳大学组织了历史上第一次正式的关于软件测试的会议。

20 世纪 80 年代早期，软件趋向大型化、高复杂度，软件的质量越来越重要，软件测试不仅是一个发现错误的过程，也成为软件质量保证的主要职能，包含软件质量评价的内容。软件开发人员和测试人员开始一起探讨软件工程和测试问题。软件测试也不再是一个一次性的、只处于开发后期的活动，而是与整个开发流程融合成一体。软件测试已成为一个专业，需要专门人才和专家运用专门的方法和手段来执行。

20 世纪 90 年代，软件行业迅猛发展，软件的规模变得非常大，在一些大型软件开发过程中，测试活动需要花费大量的时间和成本，而当时几乎完全是手工测试，测试的效率非常低，同时，随着软件复杂度的提高，出现了很多通过手工方式无法完成测试的情况。为了解决上述问题，测试工具开始出现并逐渐盛行，测试工具的选择和推广也越来越受到

重视。测试过程中,通过使用工具进行部分的测试设计、实现、执行、比较等工作,提高了软件测试的自动化程度及测试效率。

近 20 年来,随着计算机和软件技术的飞速发展,软件测试技术研究也取得了很大的突破,测试专家总结了测试模型,如著名的 V 模型、W 模型等,在测试过程改进方面提出了 TMM(Testing Maturity Model)的概念,在单元测试、自动化测试、负载压力测试以及测试管理等方面涌现了大量优秀的软件测试工具。

近几年软件测试取得了较大的发展,但仍落后于软件开发的发展水平,使得软件测试面临着很大的挑战。

(1) 测试人才缺乏。我国软件产业已经获得了长足的进步,但测试人才缺乏很大程度上制约了软件产业的发展,因此加紧建立和健全软件测试人才培养体系成为当务之急。

(2) 软件测试理论不成熟。软件测试行业的兴起将有很大程度取决于测试理论的成熟度,目前,软件测试过程中还存在一些问题没有定论或没有明确的定论,如软件测试的终止标准、如何评价测试价值等。

(3) 测试技术有待提高。目前国内软件测试技术比较落后,手工测试比重较大,自动化的性能测试、白盒测试、代码测试、安全测试等都处于初级阶段,软件测试的质量、进度、成本和风险都未得到有效的保证和控制。

5.2 什么是软件测试

1979 年,Glenford J. Myers 提出了他对软件测试的定义:"测试是为发现错误而执行的一个程序或系统的过程。"他认为软件测试的目的包括以下几点:

① 测试是程序的执行过程,目的在于发现错误。

② 测试是为了证明程序有错,而不是证明程序无错误。

③ 一个好的测试用例在于能发现至今未发现的错误。

④ 一个成功的测试是发现了至今未发现的错误的测试。

1983 年,Bill Hetzelt 在《软件测试完全指南》(*Complete Guide of Software Testing*)一书中指出:"测试是以评价一个程序或者系统属性为目标的任何一种活动。测试是对软件质量的度量。",表明软件测试的目的不仅仅是为了发现软件缺陷与错误,同时也对软件进行度量和评估,提高软件的质量。

现对软件测试的目的总结为以下三点:

① 以最少的人力、物力、时间找出软件中潜在的各种错误和缺陷,通过修正错误和缺陷提高软件质量,回避潜在的软件错误和缺陷给软件造成的商业风险。

② 通过分析测试过程中发现的问题可以帮助发现当前开发工作所采用的软件过程的缺陷,以便进行软件过程改进;同时通过测试结果的分析整理,可修正软件开发规则,并为软件可靠性分析提供依据。

③ 评价程序或系统的属性,对软件质量进行度量和评估,以验证软件的质量满足用户的需求,为用户选择、接受软件提供有力的依据。

有关软件测试的定义其实一直没有定论,人们所看到的有关软件测试的概念多是从

软件测试的目的、作用等方面进行的客观描述。而这些描述也是一百张嘴就有一百个说法，在本书中笔者比较倾向的是以下这种表达。

软件测试不仅仅是发现错误的过程，测试软件就是在可控的预置条件下操作软件的过程，其目的是：确认软件行为符合产品规格说明、发现错误和验证软件符合用户的需求。

5.3 软件测试的原则

软件产品不同于一般的产品，有其自身独特的特点，软件过程也与一般的产品生产线有着天壤之别。这种特性决定了软件测试有自己的组织和实现方式，那么软件测试都有什么要求呢？随着人们对软件过程的持续研究和不断加深的认识，人们对软件测试的理解也与日俱深，总结出越来越多的宝贵测试经验，本节将讨论一下这些有关软件测试的箴言。

1. 软件测试是证伪而非证真

软件测试是为了发现错误而执行程序的过程，软件测试完成并不能说明软件已经不存在问题了。

2. 尽早地和不断地进行软件测试

软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。据美国一家公司统计，查出的软件错误中，属于需求分析和设计的错误约占 64%，属于代码编写错误的仅占 36%，而且有研究证明，软件开发每前进一步，发现和修复软件缺陷的代价平均要增长 10 倍，因此，软件测试不应在软件代码完成后再进行，而应在软件开发的需求分析和设计阶段就开始测试工作，编写相应的测试文档，同时，坚持在软件开发的各个阶段进行技术评审和验证，这样才能尽早发现和预防错误，以较低的代价修改错误，提高软件质量。

3. 重视无效数据和非预期使用习惯的测试

测试用例的编写不仅应当根据有效和遇到的输入情况，而且也应当根据无效和未遇到的输入情况来设计。

在软件产品中突然暴露出来的许多问题常常是当程序以某些非预期的方式运行时导致的。因此，针对各种异常情况和无效输入的测试用例，似乎比针对有效输入情况的那些用例更能发现问题，这也正是边界值测试能够保持高的缺陷检出率的一个原因。

4. 程序员应避免检查自己的程序

程序员与软件产品有直接的利益关系，有很多理由支持这个原则。测试工作需要严格的作风，客观的态度和冷静的情绪。但是心理学告诉我们，人们具有一种不愿否定自己的自然性心理，这一心理状态是做好测试的一大障碍。

5. 充分注意测试中的群集现象

测试时不要以为找到了几个错误问题就已解决，不需继续测试了。经验表明，测试后

程序中残存的错误数目与该程序中已发现的错误数目或检错率成正比。根据这个规律，应当对错误群集的程序段进行重点测试，以提高测试投资的效益。在所测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集性现象，已为许多程序的测试实践所证实。

6. 用例要定期评审，适时补充修改用例

测试用例多次重复使用后，其发现缺陷的能力会逐渐降低。为了克服这种现象，测试用例需要进行定期评审和修改，同时需要不断增加新的不同的测试用例来测试软件或系统的不同部分，从而发现潜在的更多的缺陷。

7. 应当对每一个测试结果做全面检查

这是一条最明显的原则，但常常被忽视。有些错误的征兆在输出实测结果时已经明显地出现了，但是如果不仔细全面地检查测试结果，就会使这些缺陷或错误被遗漏掉。所以必须对预期的输出结果明确定义，对实测的结果仔细分析检查。

8. 测试现场保护和资料归档

出现问题时要保护好现场，并记录足够的测试信息，以备缺陷能够复现。

妥善保存测试计划，测试用例，出错统计和最终分析报告，为以后产品的升级测试提供足够的价值信息。

9. 软件测试的经济性原则

软件测试是保证软件质量的一个重要环节，其目的是找出软件中尽可能多的缺陷，但是穷尽测试又是不可能的。所以在实际项目中，考虑时间、费用、人员等因素，软件测试应该适可而止。因此，软件测试要充分利用有限的人力和物力资源，在保证软件质量的同时，遵守软件“经济性”的原则，根据程序的重要性及故障发生的损害程度来确定测试优先级；做好测试策略，使用尽可能少的测试用例发现尽可能多的软件缺陷。

5.4 软件测试的分类

软件测试种类很多，从不同角度考虑会有不用的划分方法，这里列举了几种常见的分类方法及相应的软件测试类型，如表 5-1 所示。

表 5-1 常见的软件测试类型及其分类标准

分类依据	测试类型	测试类型描述
软件开发阶段	单元测试	单元测试又称为模块测试，是对软件中最小可测试单元进行检查和验证。单元测试需要掌握软件内部设计和编码的细节知识，往往需要开发测试驱动模块和桩模块来辅助完成，一般由开发人员来执行测试
	集成测试	单元测试的下一个阶段就是集成测试，又称为组装测试，是在单元测试的基础上，将单元模块组装成系统或子系统过程中所进行的测试，重点检查软件不同单元或部件之间的接口是否正确

续表

分类依据		测试类型	测试类型描述
软件开发阶段		系统测试	系统测试是对整个基于计算机的系统的测试，软件作为计算机系统的一个元素与计算机硬件、外设、网络、支持软件、数据和人员等其他系统元素结合在一起，在真实或模拟运行环境下进行一系列的组装测试和确认测试，检查软件是否能与硬件、外设、网络、支持软件等正确配置、连接，并满足用户需求
		验收测试	验收测试是按照项目任务书或合同、供需双方约定的验收依据文档，在用户指定的或真实的环境中，对整个系统进行的测试与评审，作为用户接受或拒绝系统的依据，是软件在投入使用之前的最后测试
测试方法与技术	是否执行软件	静态测试	静态测试是指不运行被测软件本身，仅通过人工分析或检查软件的需求说明书、设计说明书以及源程序的文法、结构、过程、接口等来验证软件正确性的测试过程
		动态测试	与静态测试相反，动态测试需要运行被测软件，通过人工或工具运行软件，比较软件运行的外部表现与预期结果的差异，来验证软件的正确性，并分析软件运行效率和健壮性等性能
	是否了解内部结构	黑盒测试	黑盒测试又称为功能测试或数据驱动测试，是指不基于内部设计和代码的任何知识，而基于需求和功能性的测试，检查软件功能是否按照需求规格说明书的规定正确实现
		白盒测试	白盒测试又称为结构测试或逻辑驱动测试，是指基于代码的内部逻辑知识，即基于覆盖全部代码、分支、路径、条件的测试，检测软件内部动作是否按照规格说明书的规定正确实现，检验软件中的所有结构及路径是否都能按预定要求正确工作
测试实施组织		开发方测试	开发方测试是在软件开发环境下，由开发方检测与证实软件的实现是否满足软件设计说明或需求说明要求的过程
		用户测试	用户测试是在用户的应用环境下，用户通过运行和使用软件，检测和核实软件实现是否符合自己预期要求的过程。用户测试的主要特点是由用户找出并记录软件应用过程中出现的缺陷和错误，并对质量进行评价
		第三方测试	第三方测试又称为独立测试，通常是在模拟用户真实应用的环境下，由在技术、管理、财务上与开发方、用户方相对独立的组织进行的软件测试
测试内容		功能测试	侧重于验证测试目标预期功能，确保满足提供所需的服务、方法或用例。针对不同测试目标（包括单元、集成单元、应用程序和系统）实施和执行此测试
		安全性测试	侧重于确保测试目标数据（或系统）只供预定好的那些参与者访问。针对各种测试目标实施并执行此测试
		接口测试	侧重于验证测试目标的数据接口的正确性和对其设计的遵循性
		容量测试	侧重于验证测试目标处理大量数据的能力，可以是输入和输出或数据库中驻留的数据

续表

分类依据	测试类型	测试类型描述
测试内容	完整性测试	侧重于评估测试目标的健壮性(防止故障)和语言、语法和资源用途的技术一致性。针对不同测试目标(包括单元和集成单元)实施并执行此测试
	结构测试	侧重于评估测试目标对其设计和形式的遵循性。通常,对支持 Web 的应用程序执行此测试,以确保连接所有链接,显示合适的内容和未孤立任何内容
	用户界面测试	侧重于验证用户与软件的交互,确保用户界面向用户提供对应用程序功能的相应访问和浏览
	负载测试	一种性能测试,侧重于验证和评估在满足性能指标的情况下,系统所能承受的最大负载量
	压力测试	一种性能测试,侧重于通过确定一个系统的瓶颈或者不能接收的性能点,来获得系统能提供的最大的服务级别
	疲劳强度测试	侧重于验证测试目标在异常或极端条件(如资源减少或用户数过多)之下其性能行为的可接受程度
	恢复性测试	侧重于应用程序或整个系统可以从各种硬件、软件或网络故障(这些故障会造成不当的数据丢失或数据完整性问题)中成功地进行故障转移和恢复
	配置测试	侧重于确保在不同的硬件和软件配置上实现预期的测试目标功能
	兼容性测试	侧重于确保受测试系统与其他软件可以共存运行
	安装测试	侧重于确保在不同硬件和软件配置上以及不同条件(如磁盘空间不足或电源中断)下按计划安装测试目标。针对应用程序和系统实施并执行此测试。

5.5 软件测试基本方法

软件测试的方法和技术是多种多样的,可从不同的角度进行划分,按照是否需要执行被测软件的角度,可分为静态测试和动态测试;从测试是否针对系统的内部结构和具体实现算法的角度来看,可分为白盒测试和黑盒测试。这里重点介绍一下白盒测试和黑盒测试。

5.5.1 黑盒测试

黑盒测试是通过测试来检测每个功能是否都能正常使用,是针对程序的外部结构对软件界面和功能进行的测试。

具体的黑盒测试用例的设计方法包括等价类划分法、边界值分析法、错误推测法、因果图法、判定表驱动法、正交试验设计法、功能图法等。

1. 等价类划分

等价类划分是将程序的输入域分成若干部分,然后从每一部分中选取少数具有代表性的数据作为测试用例。这是一种重要的、常用的黑盒测试用例测试设计方法。

使用等价类划分设计测试用例,需要考虑有效等价类、无效等价类两种情况,保证软件能够接受合法数据,也能经受意外的考验。

2. 边界值分析法

边界值分析法是对等价类分析方法的补充,是针对各种边界情况设计测试用例,应当选取正好等于、刚刚大于、刚刚小于边界的值作为测试数据,而不是选取等价类中的典型值或任意值作为测试数据。

3. 错误推测法

错误推测法是基于经验和直觉推测出程序中可能存在的错误和容易发生错误的情况,有针对性地设计测试用例的方法,例如输入数据为零、输出数据为空、输入表格为空、输入表格只有一行等情况都是容易出现错误的一些数据输入情况。

4. 因果图法

因果图法就是从程序规格说明书的描述中找出因(输入条件)和果(输出或程序状态的改变),通过因果图转换为判定表,最后为判定表中的每一列设计一个测试用例。

5. 其他黑盒测试方法

(1) 判定表驱动法。判定表驱动法是利用判定表设计测试用例的方法,所谓判定表是分析和表达多逻辑条件下执行不同操作的情况的工具。

(2) 正交试验设计法。正交试验设计法是使用已设计好的正交表格来安排试验并进行数据分析的一种方法,目的是用最少的测试用例达到最高的测试覆盖率。

(3) 场景法。场景法是基于用户故事的用例设计方法,场景法生动得再现了用户实际应用场景中事件发生时的情景,有利于设计测试用例,同时也使测试用例更容易被理解和执行。

(4) 功能图法。功能图法是用功能图形象地表示程序的功能说明,并机械地生成功能图的测试用例。功能图测试用例由测试中经过的一系列状态和在每个状态中必须依靠输入输出数据满足的一对条件组成。

5.5.2 白盒测试

白盒测试是根据被测程序的内部结构设计测试用例的一种测试方法,白盒测试需要阅读程序源代码,采用人工或借助工具的方式对代码进行分析,以找出程序中潜在的错误,如数据类型定义错误、控制条件错误等。

白盒测试根据是否实际执行程序的角度可以划分为静态分析和动态测试,比如我们后面本书第三篇内容中介绍的代码走查、审查等均属于静态分析的范畴。另外,控制流分析、数据流分析、接口分析、表达式分析等也是常见的静态分析的内容。动态测试的常见

方法有逻辑覆盖(如语句覆盖、判定覆盖、条件覆盖、判定-条件组合覆盖)、基本路径测试、域测试、符号测试、程序插桩、程序变异等。有关这些方法的具体内容及使用将在第三篇详细说明,在此不再赘述。

5.5.3 黑盒测试与白盒测试的关系

前面简单叙述了黑盒测试和白盒测试的基本方法,这些方法加起来有十多种,为了增进理解,下面对这些方法做归类处理,并简单地分析这些方法之间的关系(见图 5-1)。

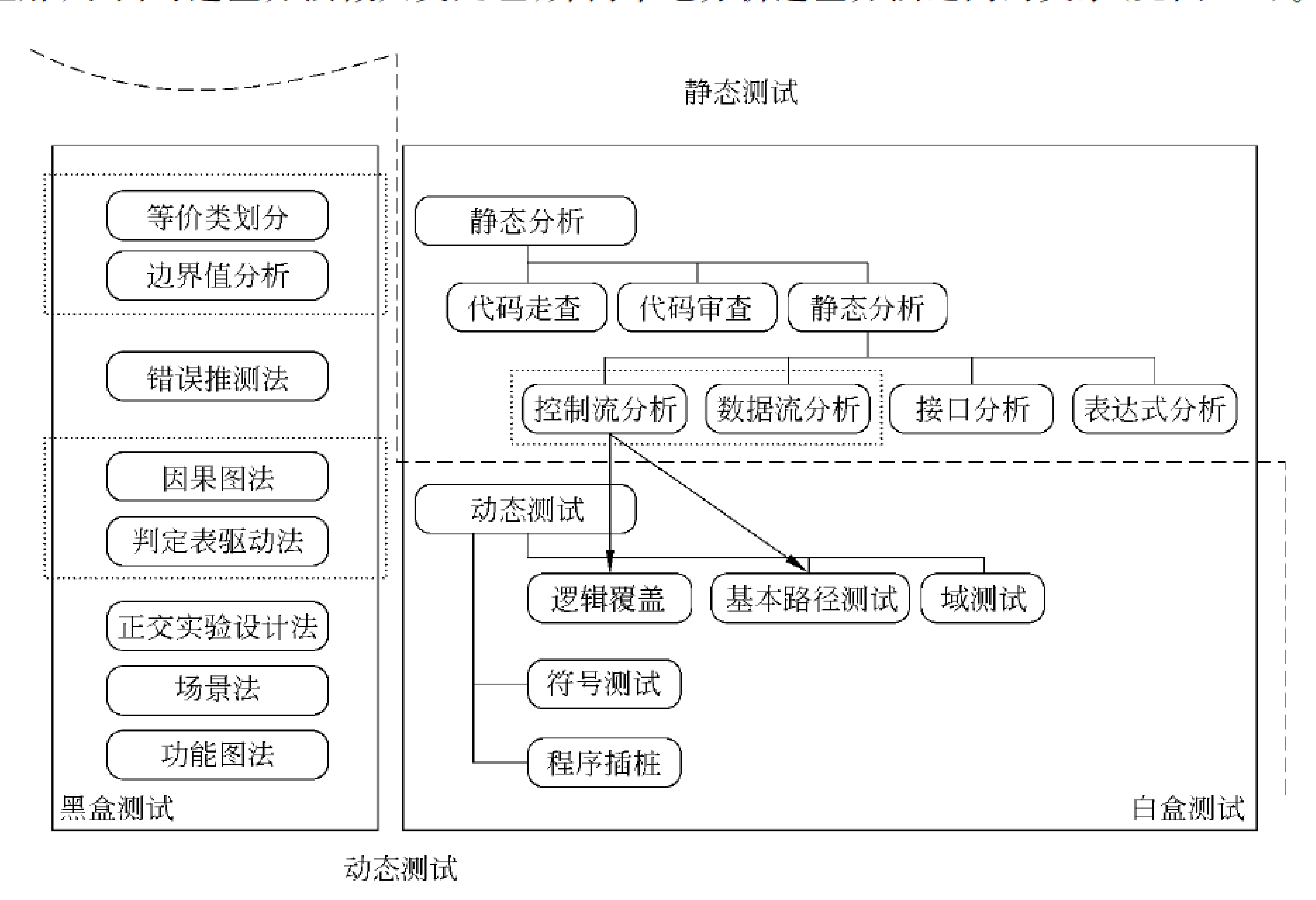


图 5-1 测试方法分类与关系分析

软件测试的基本方法是黑盒测试和白盒测试,如果从是否执行被测软件的角度区分,则白盒测试中的动态测试方法与黑盒测试共同组成动态测试部分,白盒测试的静态分析方法构成静态测试的内容。

在黑盒测试方法中,等价类划分和边界值分析关系密切,等价类的划分基点往往来源于边界分析的结果。因果图和判定表驱动也往往组合起来使用,判定表的条件正是来源于因果图中的“因”,而结果正是因果图中的“果”。因果图中的约束关系有利于消除判定表中的伪测试用例。

在白盒测试方法中,静态分析和动态测试也并不是因为有了这种形式上的区分,而“老死不相往来”,事实上静态分析的结果正是某些动态测试方法设计和实施的基础。例如,在进行判定/条件覆盖测试和基本路径测试时,首先需要对模块的控制结构进行分析,也就是控制流分析。

本章小结

在理解第 4 章中“软件质量”的概念后，本章正式引入软件测试的概念，软件测试是保证软件质量的有效手段。

本章从软件测试基础着手，着重讲解了软件测试发展历程、概念及发展现状、软件测试分类及基本方法等内容，让读者快速理解软件测试的基础知识，建立软件测试的理论认识。

CHAPTER

第 6 章 软件测试过程模型

目前已经有很多的软件开发模型，如瀑布模型、原型模型、增量模型、渐进模型、快速软件开发（Rapid Application Develop，RAD）以及 Rational 统一模型（Rational Unified Process，RUP）等，这些模型能够清晰、直观地表达软件开发过程，明确规定软件开发的主要活动和任务，对软件开发过程有很好的指导作用。软件测试是与软件开发类似，同样是一系列有计划的活动，同样也需要相关模型来指导实践，因此产生了软件测试过程模型。

6.1 什么是软件测试过程模型

软件测试过程模型是软件测试的工作框架，用于指导软件测试过程，在软件测试过程中使用合理的测试模型可以降低成本，提高效率。常见的软件过程模型包括 V 模型、W 模型、X 模型、前置测试模型、H 模型等，这些模型描述了软件测试过程，同时对软件开发过程进行了总结，体现了软件测试与开发的融合。

6.2 常见的软件测试过程模型

6.2.1 V 模型

V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，在英国国家计算中心文献中发布，旨在缩短产品的开发周期，提高开发速度。

V 模型主要反映测试活动与分析、设计的关系，如模型图（见图 6-1）所示，图中箭头代表了时间方向，从左到右，描述了基本的开发过程和测试行为，明确标明了不同级别的软件测试过程，并清楚地描述了不同测试阶段与开发过程各阶段的对应关系。

V 模型指出，单元测试和集成测试是验证程序详细设计及概要设计，检测程序的执行是否满足软件设计要求；系统测试是验证系统设计，检测系统功能、性能是否达到系统设计的指标；验收测试追溯软件需求说明进行测试，确定软件的实现是否满足用户需求和

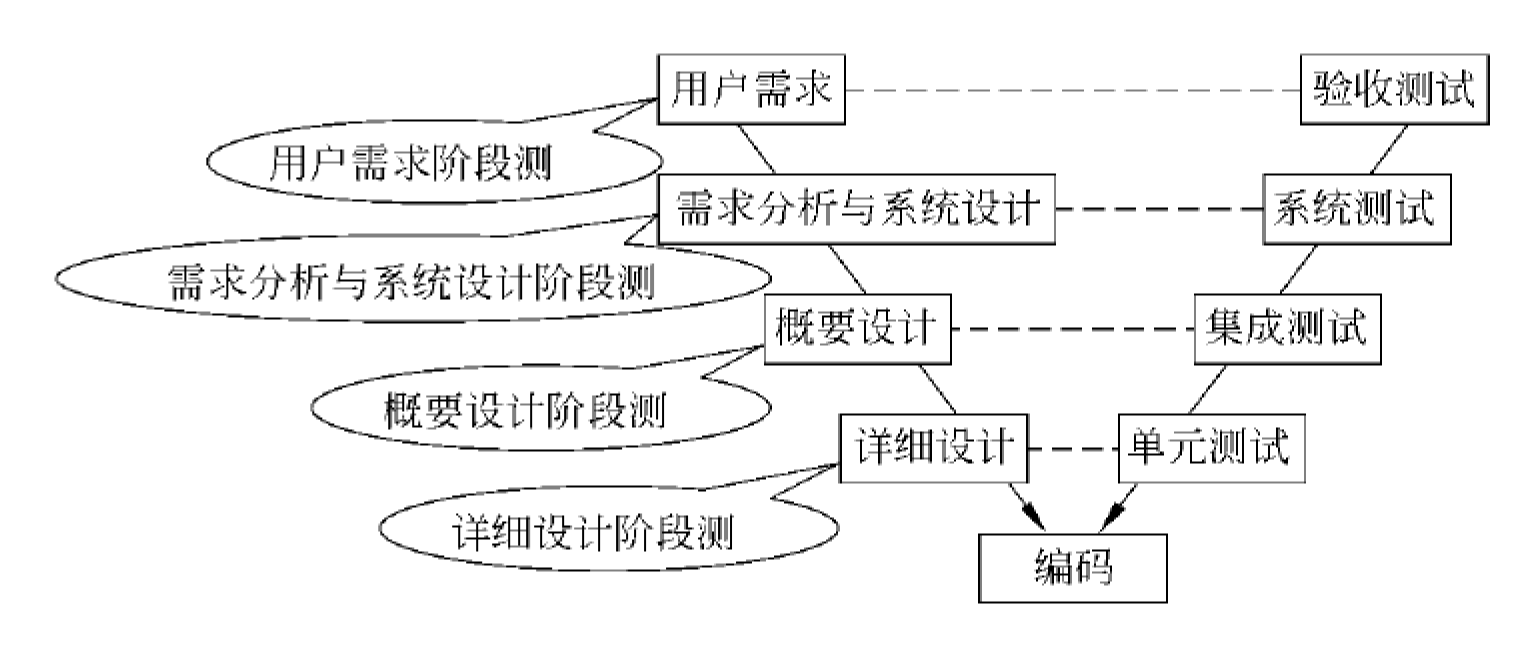


图 6-1　软件测试 V 模型

合同的要求。

V 模型没有明确说明早期的测试,将软件测试过程作为在需求分析、系统设计及编码之后的一个阶段,忽略了软件测试对需求分析、程序设计验证的意义,导致需求分析及设计阶段中隐藏的问题到后期才能被发现。

6.2.2　W 模型

W 模型由 Evolutif 公司提出,它是在 V 模型的基础上增加了软件测试与开发同步进行的过程,体现了“尽早地和不断地进行软件测试”这一原则。

通过 W 模型图(见图 6-2),可以看出,软件测试是伴随这个开发周期的,当相应的开发活动完成,其相应的测试便可以开始执行,且测试对象包括程序、需求、功能、设计等。

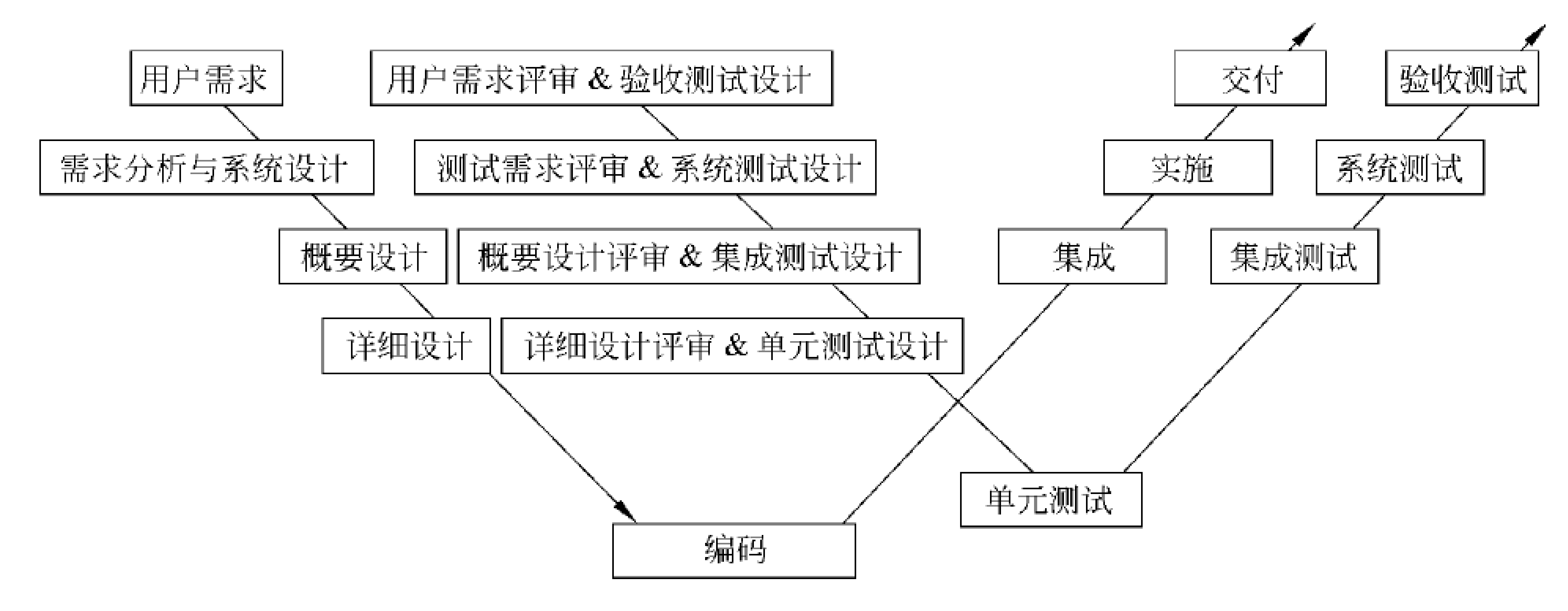


图 6-2　软件测试 W 模型

相对于 V 模型,W 模型更科学,强调了测试与开发是同步进行的,有利于尽早发现问题。但 W 模型和 V 模型都把软件开发视为需求、设计、编码等一系列的串行活动,同时,开发和测试保持着一种线性的前后关系,只有上一阶段完成后,才可以开始下一阶段的活动,不能支持迭代、自发性以及变更调整。

6.2.3　X 模型

X 模型的基本思想是由 Marick 提出的。Marick 对 V 模型的最主要批评是 V 模型

无法引导项目的全部过程，他认为一个模型必须能处理开发的所有方面，包括交接，频繁重复的集成，以及需求文档的缺乏等，但首先 Marick 不建议要建立一个替代模型。Robin F. Goldsmith 引用了 Marick 思想，并重新组织，形成了 X 模型。

X 模型图(见图 6-3)左边描述的是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁的交接，通过集成最终合成为可执行的程序，这一点在图的右上部分得以体现，这些可执行程序还需要进行测试，已通过集成测试的成品可以进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。

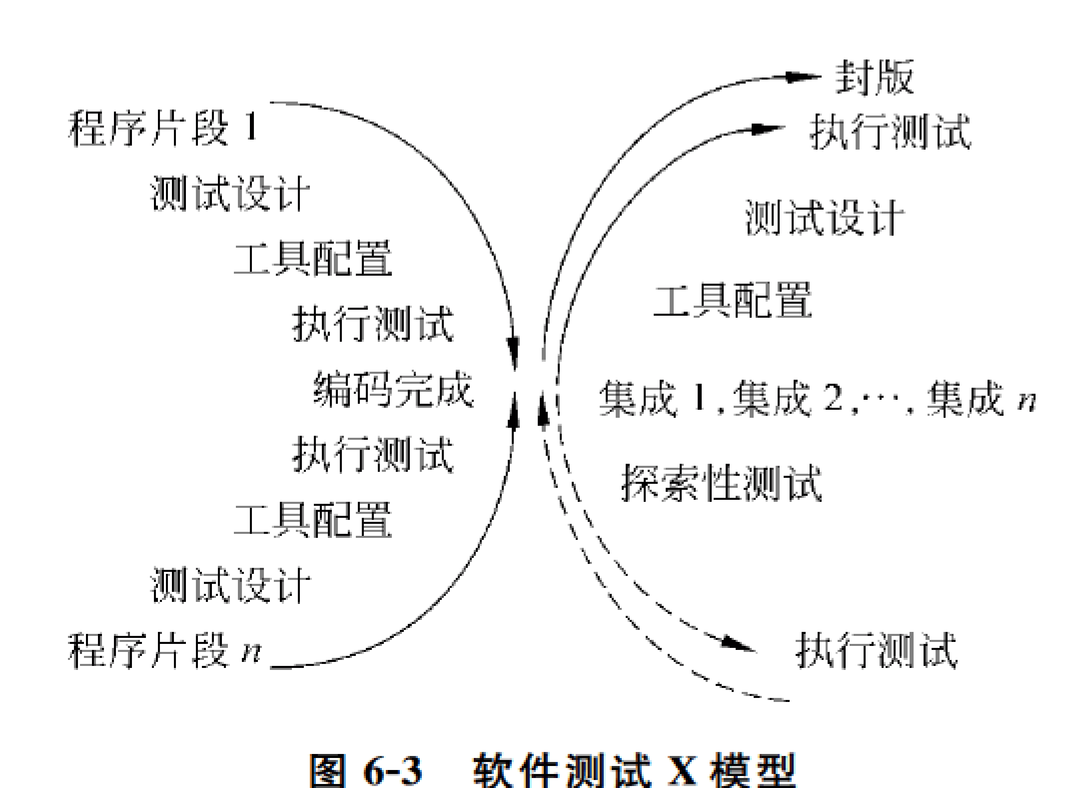


图 6-3　软件测试 X 模型

由图 6-3 可见，X 模型还定位了探索性测试，这是不进行事先计划的特殊类型的测试，只是测试人员的随机测试，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误。

另外，X 模型不要求对每一个程序片段都进行单元测试，允许跳过单元测试直接进行集成测试，但未提供跳过单元测试的判断准则。

X 模型填补了 V 模型和 W 模型的缺陷，为测试人员和开发人员带来明显的帮助。

6.2.4 前置测试模型

前置测试模型是由 Robin F. Goldsmith 等人提出，是一个将测试和开发紧密结合的模型，要求对每一个交付的内容进行测试，如图 6-4 所示。

前置测试模型体现了以下特点：

① 开发和测试相结合。前置测试模型将开发和测试的生命周期整合在一起，标识了项目生命周期从开始到结束之间的关键行为，并且表示了这些行为在项目周期中的价值所在；另外，前置测试将测试执行和开发结合在一起，并在开发阶段以编码-测试-编码-测试的方式来体现，即程序片段编写完成即可进行测试。

② 对每一个交付内容进行测试。每一个交付的开发结果都必须通过一定的方式进行测试。源程序代码并不是唯一需要测试的内容。在图 6-4 中的椭圆框表示了其他一些要测试的对象，包括可行性报告、业务需求说明，以及系统设计文档等。

③ 包括两项测试计划技术。一是开发基于需求的测试用例，这为以后程序测试做好

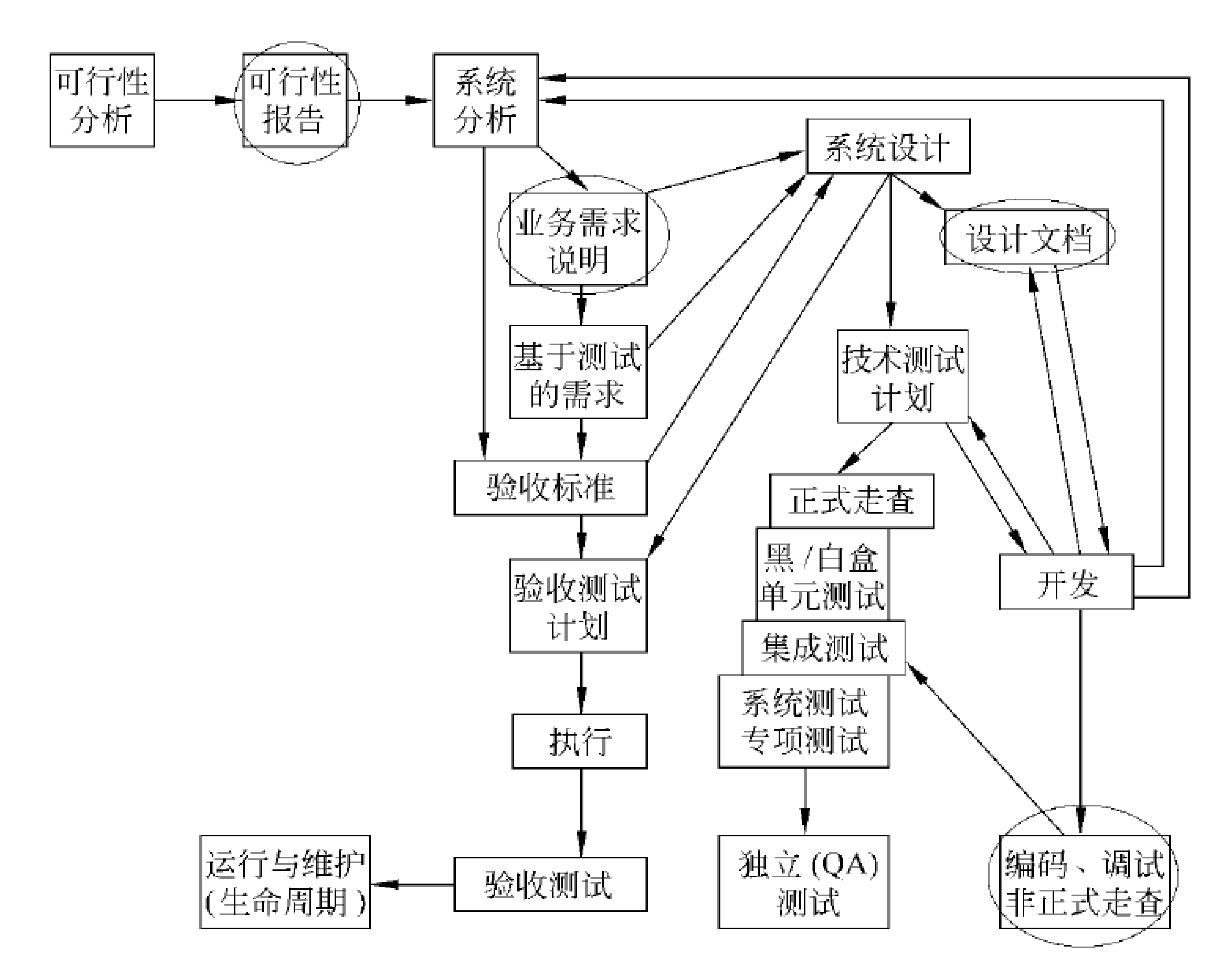


图 6-4　软件前置测试模型图

初始化准备，同时验证了需求是否可测试；二是定义验收标准，用于后期的交付验收，另外，在前置测试之前定义验收标准，有助于检查需求的正确性及完整性。

④ 设计阶段进行测试计划和设计。设计阶段是做测试计划和设计的最好时机，不做测试计划和设计，或测试开始前才开始进行测试计划和设计，易导致测试只是验证了程序正确性，而不是验证整个系统本该实现的全部内容。

⑤ 验收测试和技术测试保持相互独立。前置测试模型提倡验收测试和技术测试沿两条不同的路线进行，每条路线分别验证系统是否能够如预期一样正常工作，这样可以提供双重保险，保证设计及程序编码能够符合最终用户的需求。验收测试既可以在实施阶段的第一步来执行，也可以在开发阶段的最后一步执行，当单独设计好的验收测试完成了系统的验证，即可认为这是一个正确的系统。

⑥ 反复交替的过程。依据项目中存在很多的变更与反复的情况，前置测试模型对反复和交替进行了非常明确的描述。

6.2.5　H 模型

为更好地体现测试流程的完整性，提出 H 模型，它形成一个有组织、结构化的独立流程，强调测试准备和测试执行分离进行。

见图 6-5 演示了在整个生产周期中某层次上的一次测试“微循环”，图中“其他流程”可以是任意开发流程，非开发流程，甚至测试流程本身。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

H 模型揭示了：

① 软件测试不仅指测试执行，还包括很多其他活动。

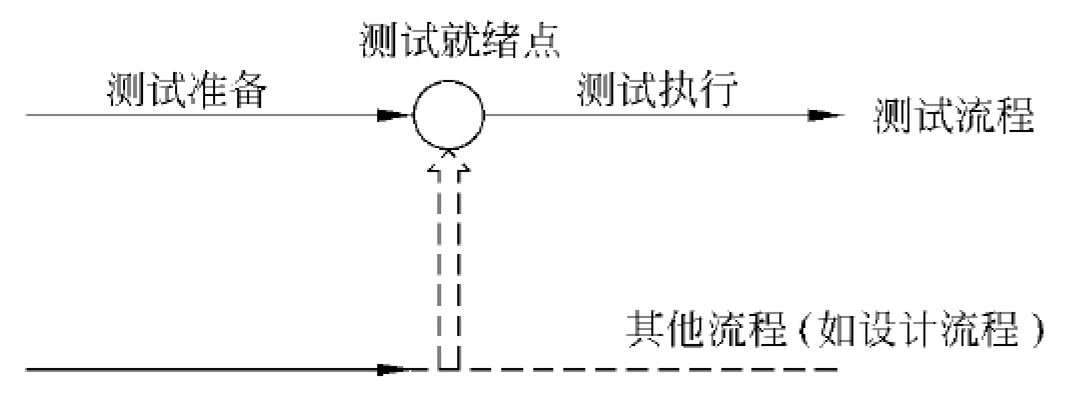


图 6-5　软件测试 H 模型

② 软件测试是一个独立的流程,贯穿产品整个生命周期,与其他流程并发地进行。

③ 软件测试要尽早准备,尽早执行。

④ 软件测试是根据被测物的不同而分层次进行的。不同层次的测试活动可以是按照某种次序先后进行,也可以是反复进行的。

6.2.6　软件测试模型比较

前面已经对软件测试模型进行了介绍,这里汇总了 V 模型、W 模型、X 模型、前置测试模型、H 模型的特点,以便读者更好地理解各类测试模型。

表 6-1　软件测试模型横向分析表

测试模型	特　　点
V 模型	V 模型强调软件开发过程中需要经历若干个测试级别,并将每个测试级别与开发级别相对应。但该模型把软件测试作为编码完成之后的活动,忽略了需求分析与设计等前期阶段的测试,导致部分错误直到后期才能发现
W 模型	W 模型补充了 V 模型的不足,强调了软件开发过程与测试过程并行进行。但该模型仍把软件开发视为串行活动,同时,开发和测试保持着线性的前后关系,不能支持迭代、自发性以及变更调整
X 模型	X 模型是对 V 模型的改进,该模型提出针对单独的程序片段进行相互分离的编码和测试,此后通过频繁的交接,通过集成最终合成为可执行的程序。X 模型并不要求对每一个程序片段都进行单元测试,但模型没有提供是否要跳过单元测试的判断准则
前置测试模型	前置测试模型结合了 V 模型 W 模型的特点,将测试和开发紧密结合,标识了项目生命周期从开始到结束之间的关键行为,要求对每一个交付的内容进行测试
H 模型	H 模型强调测试是独立的,只要测试准备完成,就可以执行测试,体现了测试流程的完整性

本章小结

软件测试过程模型是软件测试的工作框架,对软件测试过程有很好的指导意义。本章详细介绍了 V 模型、W 模型、X 模型、前置测试模型、H 模型等常见过程模型, V 模型强调了在整个软件项目开发中需要经历的若干个测试级别,并与每一个开发级别对应,但

忽略了测试的对象不应该仅仅包括程序，没有明确指出对需求、设计的测试；W 模型补充了 V 模型中忽略的内容，强调了测试计划等工作的先行和对系统需求和系统设计的测试，与 V 模型相同，没有对软件测试的流程进行说明；前置测试模型是一个将测试和开发紧密结合的模型，此模型将开发和测试的生命周期整合在一起，要求对每一个交付内容进行测试；H 模型强调测试是独立的，只要测试准备完成，就可以执行测试。

CHAPTER

第 7 章

软件测试生命周期

软件测试是一项重要且复杂的工作，相对于软件开发生命周期(SDLC)的每个阶段而言，软件测试也有自己的生命周期，在软件测试生命周期里，最基本的需求是对于软件测试的控制和处理——手工测试、自动化测试以及性能测试(见表 7-1)。

表 7-1　软件测试生命周期

阶　　段	活　　动	输 出 产 物
测试计划	创建一个总体的测试计划，对软件测试活动作出整体规划	测试计划、风险评估表、测试策略
测试分析	对被测软件系统的测试需求进行采集和分析，提炼软件的质量需求，选择相适应的测试类型，形成测试需求跟踪矩阵，完善测试计划	修订完善的测试计划、测试需求跟踪矩阵
测试设计	设计和完善测试用例，决定哪些用例自动化执行，哪些手工执行	测试用例、测试数据
	自动化测试脚本编制	测试规程/测试脚本、驱动器
测试执行	按照测试计划要求执行测试并进行回归测试	测试结果、缺陷报告
测试评估	评估测试过程，编写测试报告	测试执行记录(测试日志、测试记录、缺陷(事件)报告和测试总结报告)；测试过程改进计划(选项)

7.1　测试计划

软件测试计划一般由经验丰富的项目负责人编写，是指导测试过程的纲领性文件，让测试人员明确测试任务及方法，保持测试实施过程的顺畅沟通，跟踪、控制测试进度，同时对开发工作、整个项目规划、项目经理审查等都有辅助性作用。

软件测试计划编写六要素：Why——测试目的、What——测试的内容、When——测试的起止时间、Where——测试环境及地点、Who——测试组成员、How——测试的步骤及方法，即 5W1H。

不同的项目，其测试计划也会有所不同，测试计划通常应清晰描述以下内容：

① 项目测试背景。单位情况、应用状况、测试目的等。

② 项目概述。被测项目的性质、类型、体系结构、应用领域、工作环境、主要功能等。

③ 测试内容及目标。确定软件测试范围和优先级，制定测试目标。

④ 测试策略。制定整体测试策略、所使用的测试技术和方法。

⑤ 资源列表。进行测试所需要的软硬件、测试工具、培训资料及要求、文档、必要的技术资源等。

⑥ 工作安排。测试过程中每个阶段的工作安排，相应的人员及时间安排等。

⑦ 测试标准。测试开始、暂停、重启、完成等标准。

⑧ 风险分析。分析测试过程中可能存在的技术、人员、资源、进度等方面的风险及对策。

7.2 测试分析

任何一个项目测试执行之前，必须要了解项目的功能、业务流程、测试规模、复杂程度等内容，而这些内容只能通过需求分析得到。软件测试需求分析是测试工作的基础，其详细程度将直接影响测试设计的完整性，详细分析并了解了测试需求，才能清晰准确掌握测试内容，更好地把握测试的质量与进度。

软件测试需求通常是通过分析软件需求而形成的，但测试需求并不等同软件需求，需要从测试的角度上分析整理。软件测试需求通常存在于与软件相关的各种文档资料，如软件需求规格、合同及约定、界面设计等，另外与客户或系统分析员的沟通信息、业务背景资料、正式与非正式的培训资料等都是收集测试需求的途径。分析过程中，既要归纳出明确提出或说明的软件需求，还要分析出相应的隐含需求。

软件需求分析目的是确定软件功能、流程、非功能方面的测试内容，需求分析的内容包括以下几方面：

① 技术架构分析。主要分析软件的系统架构和所采用的实现技术。

② 功能结构分析。主要分析系统包含的功能模块并明确各模块之间的调用关系。

③ 部署环境分析。主要分析被测系统的软件、硬件、网络等方面的环境。

④ 业务流程分析。主要分析系统应用领域、总体业务流程，说明各个功能模块之间的业务顺序，和各个功能模块之间传递的信息和数据。

⑤ 功能规格分析。主要分析各功能模块中主要输入数据的规格和约束、每个具体功能的处理机制、操作顺序、输出结果。

⑥ 非功能需求分析。主要分析软件的各种非功能性需求，包括安全性、可靠性、用户文档、效率、易用性、可维护性、可移植性等方面。

7.3 测试设计

软件测试设计是软件测试中的一个重要过程，软件测试设计的详细程度将直接决定测试的质量。测试设计包括测试用例设计、测试流程安排两方面内容。

1. 测试用例设计

软件测试用例是测试执行的依据，是在测试需求分析中确定出测试内容的基础上设计形成的，其基本要素包括测试用例编号、测试标题、重要级别、测试需求、前提条件、测试输入、操作步骤、预期结果。

“测试用例应该详细记录所有的操作信息，即使一个没有接触过系统的人员也能进行测试”，完全按照这种观点去执行测试项目是合适的，实际测试过程中，可根据资源配备、人员熟悉情况、时间等情况制定不同详细程度的测试用例，但测试用例设计应掌握一些基本的原则：

① 测试用例中的输入数据、操作应能够代表并覆盖各种合理及不合理、合法及非法、边界及越界、极限等情况。

② 一个测试用例尽量只对应一个测试点。

③ 测试用例的描述应清晰、易理解，保证测试步骤的唯一性、测试结果的可判定性等。

另外，在自动化测试时使用的脚本也是一种测试用例的表达形式。测试脚本的开发需要有一定的开发基础。

2. 测试流程安排

测试流程是测试的总体思路和顺序，对测试执行的先后顺序做出整体安排。测试流程设计可以在测试要点确定后进行，也可以在具体测试方法设计完成后进行。

通常的测试流程为，测试相关准备工作→功能方面测试→整体流程的测试→非功能性方面测试，如果需要，还需对每部分测试内容的测试流程进行说明，如功能方面测试的先后顺序，先测试哪些功能的哪些部分，后测试哪个功能的哪个部分。

7.4 测试执行

根据测试设计确定出测试前期的准备工作，如测试数据准备、测试环境准备、测试工具准备、文档或其他资源的准备，保证系统的所有待测功能都可以执行，不能出现因数据的准备不足而导致系统某些功能无法执行的情况。

另外，测试执行过程中还应该注意以下几个方面：

① 如实记录测试过程。测试执行人员应详细记录测试一致情况，不一致输入、操作步骤、输出结果，以及未测试项等信息，方便整理测试问题报告；记录测试执行人员及时间，有助于测试问题的追溯及整改。

② 及时更新测试用例。测试执行过程中，应该注意及时更新测试用例。往往在测试执行过程中，才发现测试用例遗漏或不适用等情况，应及时补充、修改测试用例。

③ 问题报告要完整准确。软件问题报告是测试人员绩效的集中体现，因此，提交一份优秀的问题报告很重要。软件测试问题应包括软件配置、硬件配置、测试输入、操作步骤、输出结果、相关日志、测试结果及整改情况等。

当一个周期测试执行完成后，需将测试情况与测试目标进行比较，确定是否需第二周

期的测试，即回归测试。回归测试是针对已测试过的软件中出现的错误进行回归检查，同时对系统新功能和特征进行测试。回归测试过程中可从原设计文档中选择或修改原有测试方法和用例，或设计新的测试方法与用例，补充相应的测试数据、测试资源，建立相应的测试环境，确定相应的测试顺序。回归测试原则如下：

① 文档功能和数据中所有改变的地方都应被测试。

② 受改变部分的或受要求的系统中的改变影响的所有未改变的部分都应被测试。

③ 所有其他部分应至少按样本进行测试。

整个测试过程中，需建立缺陷跟踪与管理机制。缺陷跟踪与管理是软件测试工作的一个重要组成部分，软件测试的目的是为了尽早发现软件中的缺陷，对缺陷进行跟踪管理是确保每个被发现的缺陷都能够及时得到处理。缺陷跟踪与管理主要包括收集缺陷、缺陷统计与分析、缺陷跟踪、缺陷处理。

7.5 测试评估

测试评估就是要回顾整个软件测试过程，对局部数据进行采样分析，判断测试是否充分，是否达到了测试目标，以便编制测试报告。

测试报告是测试阶段最后的文档产出物，优秀的测试经理应该具备良好的文档编写能力，一份详细的测试报告包含足够的信息，包括产品质量和测试过程的评价，测试报告基于测试中的数据采集以及对最终的测试结果分析。

软件测试目的是发现尽可能多的缺陷，但实际执行过程中，由于时间、费用等因素，完全彻底的、全面的测试是不可能实现的。通常情况下，测试终止时，软件至少要达到三点要求：确保软件能够完成它所承诺或公布的功能；确保软件满足性能和效率的要求；确保软件是健壮的、适应用户环境。

测试终止后，相对这个项目的主要工作就基本完成了，后续工作还包括准备最终测试报告、进行项目资料归档、开展项目总结会议、测试改进等。

本章小结

软件测试包括一系列的活动，在整个软件生命周期中占有重要地位。本章按照生命周期的方法分析软件测试过程，描述软件测试从开始到结束经过的一系列准备、执行、分析过程，包括测试启动、测试计划制定、测试需求分析、测试设计、测试实施、测试周期与错误修正、测试评估等活动，让读者对软件测试过程有个整体、系统的认识与理解。

PART 3

第三篇 软件测试一般过程与方法

本篇按照软件测试生命周期模型组织编写，并在讲述理论的同时穿插项目案例的实操（包含各类测试文档的编写也要融合到这里），达到理论与实践紧密融合，不仅告诉读者“做什么”同时也回答“怎么做”。

本篇是本书的核心内容，通过项目实例，向读者展示如何把测试方法应用于具体项目中。我们在这篇中介绍的软件测试的一般过程本质上是架构于 H 模型，也就是一种独立的软件测试过程，通过 HRMIS（Human Resource Management Information System）这个项目实例详细阐释了软件测试过程，同时对测试方法在实际软件测试过程中的具体应用做了实例分析。本篇内容同时融合了业界通行和笔者在测试项目实施中的实践经验，既保持了内容的严谨性和权威性，同时也兼顾了业界现状，具有较高的可操作性和现实的指导意义。

本篇术语解释

C/S：在网络连接模式中，除对等网外，还有另一种形式的网络，即客户机/服务器网，Client/Server。在客户机/服务器网络中，服务器是网络的核心，而客户机是网络的基础，客户机依靠服务器获得所需要的网络资源，而服务器为客户机提供网络必须的资源。这里客户和服务器都是指通信中所涉及的两个应用进程（软件）。使用计算机的人是计算机的“用户”（user）而不是“客户”（client）。但在许多国外文献中，也经常把运行客户程序的机器称为 client

(这种情况下也可把 client 译为“客户机”),把运行服务器程序的机器称为 server。所以有时要根据上下文判断 client 与 server 是指软件还是硬件。

B/S:B/S 是 Browser/Server(浏览器/服务器模式)的简写,浏览器/服务器模式是随着 Internet 技术的兴起,对 C/S 结构的一种改进。在这种结构下,软件应用的业务逻辑完全在应用服务器端实现,用户表现完全在 Web 服务器实现,客户端只需要浏览器即可进行业务处理,是一种全新的软件系统构造技术。

基线(BaseLine):基线是一个很常见的术语,在配置管理和项目管理里面都能看到,而且还有很多衍生的术语,例如基线提升、基线化、基线审计等。它一般用来表示一份文档的稳定状态或者代码的一个稳定版本。任何对于基线的变更要遵循严格的变更控制流程。

负载测试(load testing):负载测试是确定在各种工作负载下系统的性能,目标是测试当负载逐渐增加时,系统组成部分的相应输出项,例如通过量、响应时间、CPU 负载、内存使用等如何决定系统的性能。

压力测试(stress testing):压力测试通过确定一个系统的瓶颈或者不能接受的性能点,来获得系统能提供的最大的服务级别的测试。

容量测试(capacity test):确定系统可处理同时在线的最大用户数。

并发用户数(concurrency user):指在某一给定时间内,某个特定点上进行会话操作的用户数。

响应时间(response time):指的是客户端发出请求到得到响应的整个过程所经历的时间。

吞吐量(throughput):指单位时间内系统处理的客户请求的数量,直接体现软件系统的性能承载能力。

资源利用率:系统资源的使用程度,如服务器的 CPU 利用率、内存利用率、磁盘利用率、网络带宽利用率等。

CHAPTER

第8章 测试计划

《礼记·中庸》中有这样一句话"凡事预则立,不预则废",这里的"预"是"计划准备"的意思,就是说不论做什么事,事先有准备,就能得到成功,不然就常常会失败。

如果说软件开发在一定范围内还可以容忍在准备不充分的条件下即可以开始编码工作的话,比如名噪一时的极限编程,那么软件测试则几乎不存在这样的可能性。软件测试是一项细致的工作,其过程的工作质量的好坏对软件测试的最终效果有着绝对的影响力,从这个角度出发,甚至有部分资深的从业人员喊出"过程胜于结果"的极端口号。我们当然不会去追随"过程胜于结果"的声音,最佳的结果和与之匹配的合理过程都是我们需要的,这两者并不相悖,而是一种水到渠成的关系,有了合理的测试过程,自然就会有一个合理的测试结果。

社会各行各业总是存在各种各样的软件应用需求,这带来了软件项目的多样性,手机软件、嵌入式软件、航空控制软件等,即便在本书中所阐述的通用性软件也存在行业的差异、规模的差异等,如何为各类软件定制合理的测试需求、测试策略以及配套合理的测试资源,都是软件测试伊始首先需要解决的问题。这就是软件测试中的"预"。

"预"有计划之意,但是测试计划有时被部分组织分解为测试准备和测试计划。测试准备这个阶段客观上说没有一个非常严格的界定,所做的事情诸如测试需求的整理分析、测试环境的构建等,从内容上来说,测试准备完全可以置入"测试计划"中,作为"测试计划"的某项计划任务去处理,这样来得更加简单清楚。所以,本章讨论的测试计划不对测试准备作出区分而统一以测试计划作为软件测试生命周期的第一个阶段。

8.1 项目启动场景

在我国的IT产业,最近十年,软件测试得到了迅速的发展。在一份最新的产业调查中,我们看到目前大约有接近50%的软件企业设立了独立的软件测试部门,从事本企业软件产品的测试。50%这个数字,刚刚好,表达了两层意思,一方面说明我们的软件测试虽然发展迅速,但是与IT产业发达国家和地区相比仍有较大差距;另一方面也表明软件测试未来仍存在很大的发展空间,软件测试职业前景广阔。

第6章对软件测试模型做了部分介绍,其中H模型强调测试过程的独立性,其最基本

的特点就是测试与开发的分离。H 模型揭示了软件测试的以下特点：

- 软件测试不单纯是测试执行过程，还包括很多其他的活动。
- 软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。
- 软件测试要尽早准备，尽早执行。
- 从微观角度看，H 模型表达了软件测试是不排斥对被测软件产品进行分解，任何一个配置项具备了测试条件后均可先行测试。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复的。

因此，H 模型得到了更多的业界认同，成为软件测试的一个发展趋势。国内 50% 的软件企业设立了独立的软件测试部门，实现了测试与开发的分离，从形式上他们都可归为 H 模型的实践者。那么，基于 H 模型的软件测试过程是如何开始的呢？

1. 软件移交

在 GB/T 8567—2006《计算机软件文档编制规范》(以下相同处简称 GB/T 8567)中关于软件移交的描述是这样的：软件产品的开发责任从一个组织转交给另一个组织的一系列活动。一般来说，前一个组织是实现初期软件开发，而后一个组织是进行软件支持。

软件移交不仅仅存在于软件开发企业和最终客户之间，在软件开发企业内部的各个组织甚至各个职责不一的工作小组间也同样存在着移交关系。

软件开发组织(可能是开发部、产品部或者一个项目组)将软件产品的一个 Build(编译发布)交给软件测试组织(可能是测试中心、品质管理部或者一个测试团队)进行测试，也可以视为是一次软件移交。只不过这种移交内嵌于软件开发企业内部，不具备相当规范程度的管理体系的企业或单位这种交接往往没有十分严格的界限，也很容易被忽略。但是，这种移交又是必需的，一次成功的移交是接收方执行后续工作的基本保障。

在软件开发企业和最终客户之间发生的移交，往往是软件验收中进行的一项活动，指在软件开发企业履行完合同规定内容后，向客户转交项目管理、开发文档、软件代码以及后续服务项目等工作产出物的一系列活动。项目移交的过程也就是软件产品正式上线运营的过程。而存在于开发企业内部的开发组织和测试组织间的移交，内容上有精简，形式上基本类同。开发组织在交付软件进行测试时，通常会按照软件测试规范将软件产品的开发类文档(如需求规格说明书、系统设计说明书、用户操作手册等)一同交付测试组织。测试组织在接收这些交付物时，应仔细检查文档的版本及变更情况，以及与所交付软件产品的一致性和符合性。图 8-1 所示的是一个简化的软件产品开发控制过程，开发人员在完成初步的编码和单元测试之后向配置管理库中提交新版本，项目经理(或者产品经理、开发经理)会同测试负责人判断是否达到了“测试准入”条件，如果满足条件，则由发布管理员负责检出新版并编译打包交付测试组进行测试。从这个图中，我们也可以看出，开发与测试之间的移交也是有一定的层次性的，即可以是最终的完整产品性的移交，有时也可能表现为项目内部软件产品配置项的移交。

软件移交是开发与测试的重要衔接点，软件移交应作为整个项目计划的一部分，使移交工作是可预期的、可控制的，从而使测试计划更加具有可执行性。

2. 资料审查

资料审查是软件移交的主要工作内容。根据 GB/T 8567 的规定，一个软件生存周期

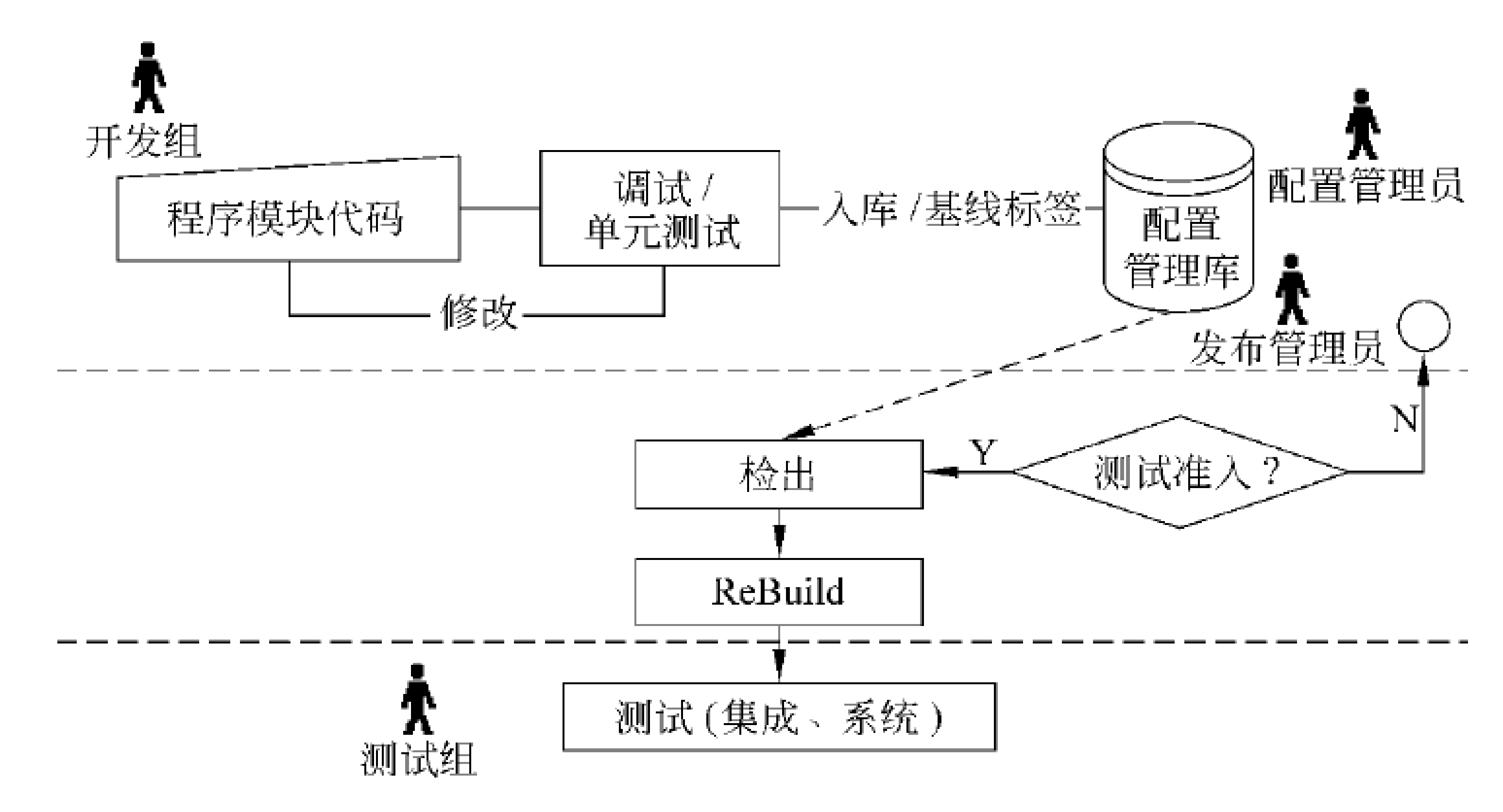


图 8-1　软件移交示意图

内要产生很多种文档，按照文档的干系人(使用人或者关联人)做了如表 8-1 所示的分类。

表 8-1　资料分类及审查要求

干系人	文档标识	文档描述	第一方、第二方	第三方
管理人员	FAR	可行性分析(研究报告)		
	—	SDP		
	SCMP	软件配置管理计划	★★☆☆☆	
	SQAP	软件质量保证计划	★★☆☆☆	
	DPMR	开发进度月报	★★★☆☆	
	PDSR	项目开发总结报告		
开发人员	—	FAR		
	SDP	项目开发计划	★★★☆☆	
	SRS	软件需求规格说明	★★★★★	★★★★★
	IRS	接口需求规格说明	★★★★★	★★★★☆
	SDD	软件(结构)设计说明	★★★★☆	★★☆☆☆
	IDD	接口设计说明书	★★★★☆	★★☆☆☆
	DBDD	数据库(顶层)设计说明	★★★★☆	★★☆☆☆
	—	TP		
	—	TSR		
测试人员	TP	测试计划		
	TCS	测试用例说明		
	TDS	测试设计说明		
	TIR	测试事件报告		

续表

干系人	文档标识	文档描述	第一方、第二方	第三方
测试人员	TITR	测试项传递报告		
	TL	测试日志		
	TPS	测试规程说明		
	TSR	测试总结报告		
维护人员	—	SRS		
	—	IRS		
	—	SDD		
	—	STR		
用户	SPS	软件产品规格说明		★★★★★
	SVD	软件版本说明		★★★★★
	SUM	用户手册		
	SOM	操作手册	★★★★☆	★★★★★

表格说明：

① 测试人员使用的文档参考了最新版 GB/T 9386—2008《计算机软件测试文档编制规范》(以下相同处简称 GB/T 9386)的有关规定。

② 根据第一方、第二方和第三方的测试目标和要求不同，对测试的输入类文档做了区分，并对文档的重要程度按照 5 星标准划分了等级。

作为软件企业内部的测试(可能是第一方或者第二方)，软件开发组织在向软件测试组织移交测试时，软件测试组织应对以下测试输入进行审查，具体包含以下资料：

- 软件需求规格说明；
- 接口需求规格说明；
- 软件结构(设计)说明；
- 接口设计说明书；
- 数据库(顶层)设计说明；
- 操作手册；
- 项目开发计划；
- 软件配置管理计划；
- 软件质量保证计划。

完整的需求规格说明有助于测试组织有效的识别测试边界，导出测试需求；详细的设计说明有助于测试组织设计充足的测试用例以达到合理的测试覆盖度；操作手册在没有足够的产品信息培训时可以帮助测试人员尽快熟悉和掌握产品的工作原理；而及时的了解项目开发计划有利于测试资源的合理分配和测试的及时性；配置管理计划指导测试人员正确的获取测试版本信息、配置项变更情况以及需要归档的测试产出物；软件质量保证计划可以指导测试人员及时了解参与各类质量保证活动，提供质量保证数据。

作为独立第三方测试，测试组织一般在接收测试样品时有着更为严格的规定，视软件

测试的介入程度不同对测试提交物的要求也有变化。通常情况下，第三方测试审查的资料包含在以下文档范围：

- 软件需求规格说明；
- 操作手册；
- 软件产品规格说明；
- 软件版本说明；
- 接口需求规格说明；
- 软件结构（设计）说明；
- 接口设计说明书；
- 数据库（顶层）设计说明。

在某些情况下，第三方测试组织可能会要求调阅有关合同文件，以对问题及争议进行进一步的确认。

事实上，对于多数测试组织来讲，在移交测试的时候能够获取到完整的文档是比较困难的，开发组织往往总是能找到各种各样的理由拒绝提供有关文档，那是因为很多不成熟的开发组织对于文档几乎失去控制，文档被完全形式化，不能及时更新开发期间做出的变更，导致文档的参考价值大幅缩水。这对于测试人员来说是一个极大的挑战。那么，如何在这样一种比较极端的工作环境下开展测试呢？

实践经验告诉我们，测试人员要善于捕捉显现的和隐含的测试需求，而这也正体现了软件质量的一个要求。什么是软件质量？软件质量就是“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。软件质量本身就是包含显性和隐含两个层面的因素的，从这个角度来说，指望软件产品的质量要求在移交测试后全部显性的落实到文档之中显然也是一个不可能实现的理想状态，所以在测试期间不能仅仅依赖于文档，文档只是测试的一个基础，除此而外，还应该善于从文档的、产品设计的等等评审会议的评审记录中，各类邮件信息中提取有益测试的信息，还有就是及时跟开发人员进行充分的沟通。

对于文档的审查是资料审查的一个重要工作，除此之外，很多资深测试人员和机构也在提倡做另一件事情，那就是“冒烟测试”。

冒烟测试（smoke test）可以理解为测试耗时短，仅用一袋烟功夫就足够了。也有人认为是形象地类比新电路板基本功能检查，任何新电路板焊好后，先通电检查，如果存在设计缺陷，电路板可能会短路，导致板子冒烟烧毁。

冒烟测试又叫做健全性检查（sanity test）或者接受测试（acceptance into test）。冒烟测试的测试对象是每一个新编译的需要正式测试的软件版本（builds），目的是确认软件核心功能、一些必要的意外处理和相对重要的功能处理可以初步正常运行，能够进行后续的正式测试工作。冒烟测试一般不需要设计专门的测试用例，通常做法是从已设计的系统测试用例集中抽取部分核心用例作为冒烟测试的测试用例集。

冒烟测试通常由一名测试人员进行，其操作方式是准备一组测试用例或者从已设计完成的测试用例集中抽取一部分在很短的时间内手动或者自动的执行一遍，以快速检验版本稳定性。冒烟测试通过与否已成为大部分企业或组织作为开发与测试移交的重要判

断依据。

软件移交和资料审查是软件产品正式从开发阶段转入测试阶段的基本工作，除此之外，对于一些规模较大的测试项目，测试团队的管理人员（测试小组组长、测试中心负责人）应适时向有关人员做出一些必要的测试说明。这些说明主要包括：

- 测试任务说明；
- 软件项目总体计划；
- 测试投入；
- 时间安排。

其组织形式可以是一次小型的项目启动会议，尤其是对于第三方软件测试，项目启动会议的作用更有不可替代的意义。项目启动会议是一个项目的开始，因此其对于测试工作的顺利开展非常重要。

对于测试团队内部的项目启动会议主要目的是让测试团队的每位成员对项目整体情况（包括项目建设背景、项目总体规划及产品要求、项目团队成员及其岗位职责以及前述的测试任务说明、测试资源投入等信息）和各自工作职责有个清晰认识，同时获得领导对项目资源承诺和保障。

对于第三方测试除了召开内部的项目启动会议以外，有时还需要召开项目外部启动会议。所谓项目外部启动会议指有项目主要干系人参加的项目启动会议。项目外部启动会议一般选择在建设方或用户现场召开。

项目外部启动会议的主要目的是让项目建设方、用户、监理方（如有）等项目主要干系方对项目整体情况（包括测试任务、测试总体方案及测试资源投入等信息）有个清晰认识，并且要让项目各主要干系方清楚各自职责和义务。让项目建设方、用户方在测试实施过程中所需要给予的支持和配合给予承诺，从而让各方就测试工作相关事宜达成一致。

8.2 测试计划

Lalitha 首倡“软件测试生命周期”这个概念，他认为软件测试不仅是运行测试，也包括设计测试和建立测试标准。有效的测试也需要好的计划、确定的过程、好的管理、清晰的文档以及测试结果的追踪。这些信息都记录在测试计划中。

软件测试是有计划、有组织和有系统的软件质量保证活动，而不是随意地、松散地、杂乱地实施过程。在 GB/T 9386 中，测试计划被定义为：描述预定测试活动的范围、方法、资源和进度的一种文档，它确定测试项、要测试的特征、测试任务、执行每一任务的人员以及需要应急对策的任何风险。

为了规范软件测试内容、方法和过程，在对软件进行测试之前，必须要制定详细的测试计划。因此项目启动之后，首先就要着手软件测试计划的制定工作。

测试计划是整个测试工作的纲领性文件，是整个测试项目的工作指南和依据。借助软件测试计划，参与测试的项目组成员，可以明确测试任务和测试方法，保持测试实施过程的顺畅沟通，跟踪和控制测试进度，应对测试过程中的各种变更。

测试计划具有以下特点。

（1）测试计划的层次性。测试计划具有层次性，通常在项目初期，限于对项目的认知深度，很难做出一份精确的测试计划。比如，在项目初期，项目主要实施的工作是需求采集分析、业务建模，对于各个模块的复杂度和测试需要的支撑资源都是比较模糊的，此时所做的测试计划只更多的是一种纲领性的测试规划，主要内容涉及项目的背景、测试范围、投入的测试资源（人力资源和时间安排）、测试需要协调和配合层面的描述、测试的工作流程、测试准入准出条件等。它对应的是项目总体开发计划。

这样的一份测试计划显然不具有太多的可执行性，改进是必然的。随着项目的进展，各个子项目的应用模型和设计基本成型，此时针对每一个子项应该独立制定测试计划，并对测试策略、方法等作更详细的描述和规定。各个子项的测试计划聚合之后形成具有更高可执行性的新的测试计划。

每一个项目都会经历单元开发、单元的集成、子系统的集成，最终形成目标系统，与之对应软件测试也会经历单元测试、集成测试、系统测试这样一个过程。测试计划的层次也表现在这里，一个完整的测试应包括单元测试计划、集成测试计划以及系统测试计划等（如图 8-2 所示）。

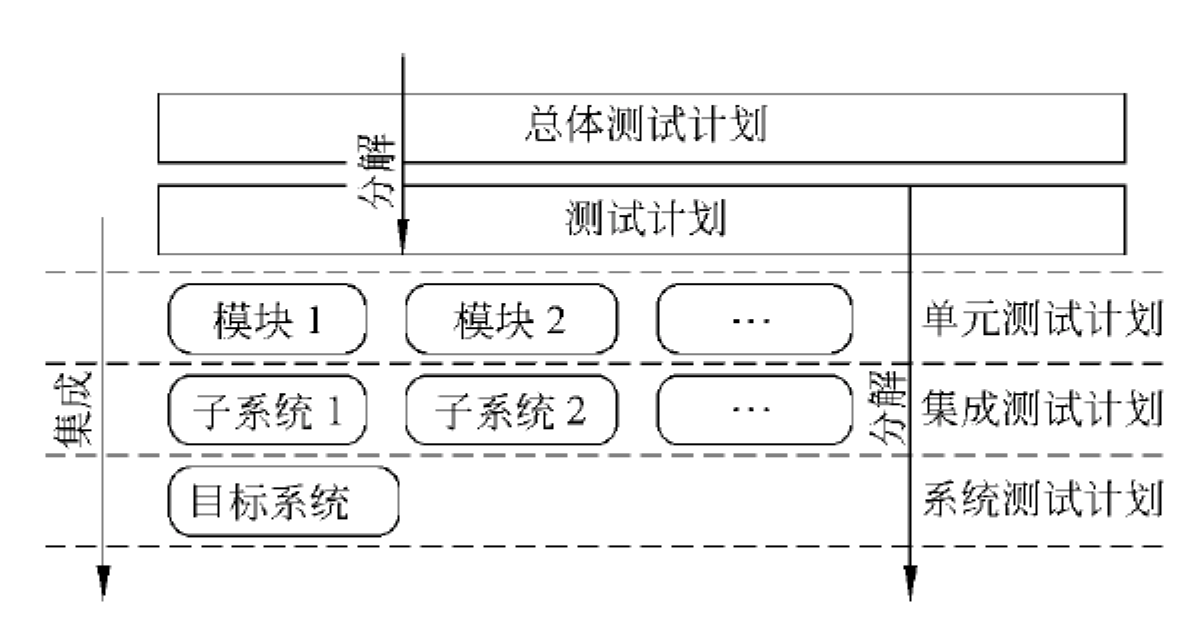


图 8-2　测试计划的层次

（2）测试计划变更频繁。软件项目是一个脑力劳动集中的工作，实施复杂度高。项目的开发过程不是一帆风顺的，经常会出现各式各样的技术问题导致开发任务的迟滞，导致工期延后，当然也有的任务会提前完成，而使后续任务得以提前。为了应对这些变化，测试计划要及时做出调整，以合理地分配测试资源，保证测试计划的顺利执行。

8.3　测试计划的编制过程及要素

如何来编制一份好的测试计划呢？有人提出 5W1H 的概念，5W1H 分析法也叫六何分析法，是一种思考方法，也可以说是一种创造技法，是对选定的项目、工序或操作，都要从原因（何因）、对象（何事）、地点（何地）、时间（何时）、人员（何人）、方法（何法）等 6 个方面提出问题进行思考。

5W1H 应用到软件测试领域，它们所代表的意思可以简要表示如下：

- Why：为什么要进行这些测试；
- What：测试哪些方面，不同阶段的工作内容；

- When：测试不同阶段的起止时间；
- Where：文档，缺陷的存放位置，测试环境等；
- Who：项目有关人员组成，安排哪些测试人员进行测试；
- How：如何去做，使用哪些测试工具以及测试方法进行测试。

5W1H 作为一种简单有效的思维模式，在很多领域得到广泛的应用，但是延伸到软件测试领域，有时要把这“六何”搞清楚却并非一件十分容易的事情。一般来说，一份测试计划的形成大体要经历以下步骤：

① 收集、研究、制定测试策略、方法；收集、研究、记录测试子项的内在联系及工作机理。

② 讨论并记录测试子项与整体项目的协作关系。

③ 定稿，并将计划的详细信息和其他必要信息文档化，例如测试子项本身的风险、测试目标、项目期限等；列出测试计划所引用或参考的文件；简要的写出测试子项的执行概要。

④ 测试计划形成以后，首先要向测试团队分发以进行内部评审，内部评审后，可以继续分发给其他干系人（即所有利益相关者和参与者）以获得更广泛全面的意见。计划制定者要汇总这些评审意见以对测试计划进行修订，并视需要重复步骤①～步骤③。及时评估计划执行过程中因为某些事务的迟滞或其他突发事件引起的日程和预算的变更，以获得管理层面的理解和支持。

⑤ 及时获取测试计划执行情况和测试资源的投入和使用情况，适时修订原有的评估计划日程和预算。如果这导致时间安排上的失误，或者除了迟滞和突发事件之外预算的增加，应上报管理层进行裁决。

⑥ 将测试计划纳入项目库或配置管理系统，对文件变更进行控制。

测试计划的编制有章可循，也有很多成型的各式模板作为参考，写一份测试计划并不是一件十分为难的事。但是反过来说，要做出一份高质量的测试计划又不是很容易的，日常工作中经常听到同事抱怨，他们认为写测试计划到底有什么用，都是些形式化的东西，跟写“八股文”似的十分枯燥，而对于开发组织似乎也没有多少参考价值。很显然，这种认识有着很大的误区，究其原因，可认为最大的问题是他们大多数时候是为了测试计划而写测试计划，写测试计划纯粹是为了应付 SQA，应付客户，这样的测试计划其应用价值显然会大幅缩水，也因为文档的价值感缺失而造成编制人员的厌倦情绪。

应该说，测试计划的形成本质上是对一次测试任务逐步认知和分析的过程，其间主要的活动可以概括为如图 8-3 所示的控制流程。每一次测试分析动作会对应的产生一组数据资料，将这些数据资料汇总整理作为测试计划的输入项，测试计划也就初步形成了，是一种水到渠成的关系。

8.3.1 测试的质量需求

软件测试是服务于软件质量的，测试从业人员作为基本的职业觉悟应牢记这一点，当然对于大部分初入本行业的人员理解这一点又是不容易的，但希望测试人员都能牢记这一点并逐步的去理解，这对于你是否真的能做好软件测试至关重要。

最终用户对于软件质量需求是测试的出发点，也是对软件质量进行评价的唯一标准。

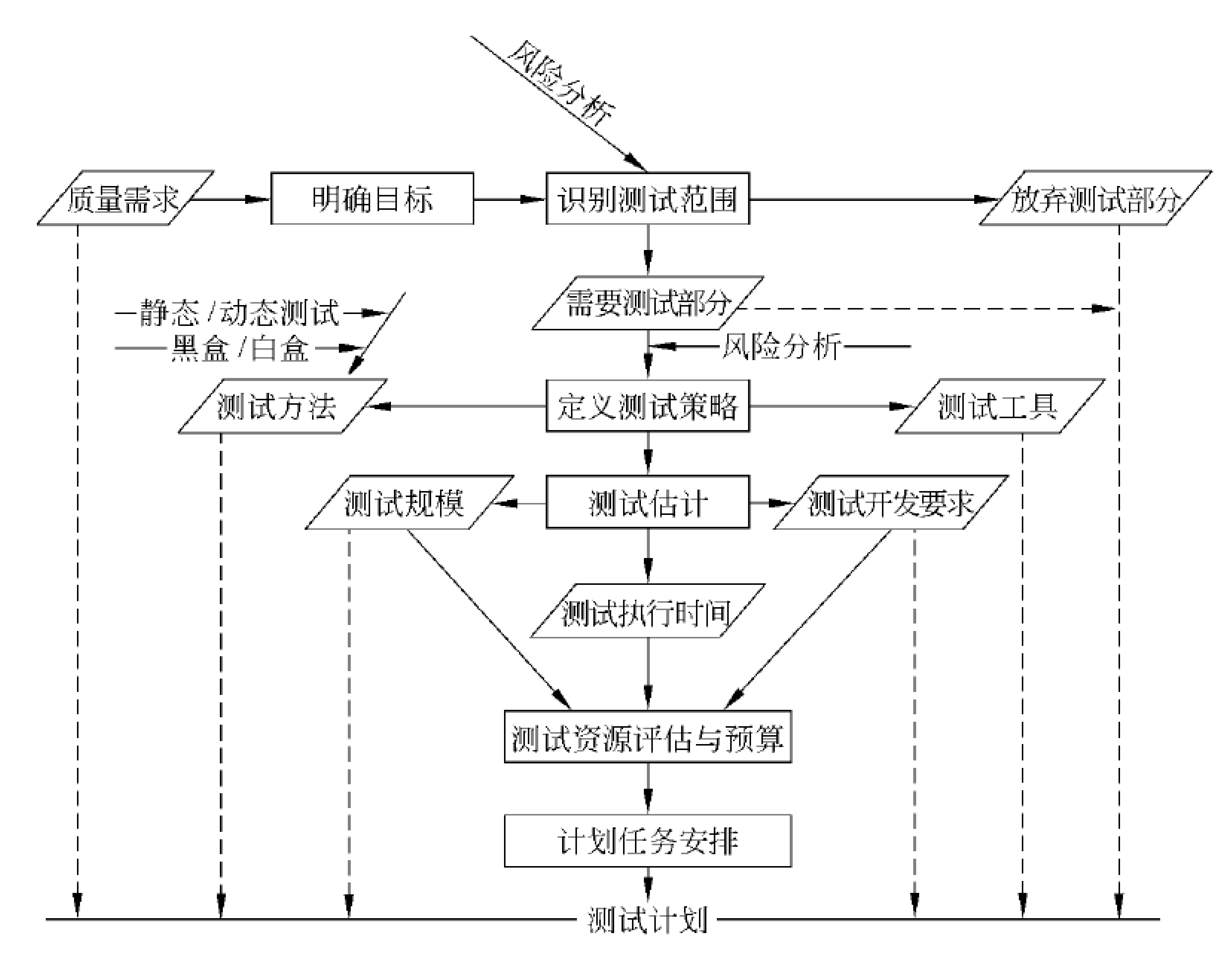


图 8-3 测试计划的编写过程

质量需求由功能性需求和非功能性需求构成，功能性需求比较好理解，而非功能性需求一般来说可以从性能、可靠性、可移植性方面去发掘。

【案例项目 HRMIS 测试质量需求】

分类	内容描述			遗留问题
结构	易用性、稳定性、扩展性			
	界面风格保持统一			
功能	管理功能	系统访问权限管理		
		用户管理		
		部门管理		
	业务功能	员工基本信息管理	基本信息管理	
			工作履历管理	
			培训实施记录	
			合同管理	
			部门调动	
			信息变更历史记录	
		考勤/考评管理		
		员工工资管理		
		培训管理	培训信息管理	
			培训机构管理	
	接口	考勤系统		接口需求及设计
		财务管理系统		接口需求及设计
		物流管理系统		接口需求及设计
		生产管理系统		接口需求及设计
		CRM		接口需求及设计
性能	系统登录	15User并发，平均响应时间<10s		
	考勤高峰	15User并发，平均响应时间<10s		
	信息修改	平均小于15s		确定典型业务
	数据转换和传送	平均小于15s		确定典型业务
其他要求	不要求跨平台			
	故障恢复			
	涉及时间的要求精确到日期，不要求精确到时间			
	培训费用精确到元角分，计费币种为人民币			

说明：粗体字部分是跨出原系统需求的新增测试需求；阴影标注区是需要进一步确认的测试需求。

确定需要测试的软件特性。根据软件开发合同或系统/子系统设计文档的描述确定系统的功能、性能、状态、接口、数据结构、设计约束等内容和要求，对其标识。若需要，将其分类，并从中确定需测试的软件特性。

8.3.2 风险分析

风险分析是一个定义并且找出阻止潜在的问题变成现实的方法的过程。逐项列出能够影响整个项目成败的关键问题、技术难点和风险，指出这些问题对项目的影响，包括那些对系统任务最关键和那些帮助降低威胁系统成功运行的最大风险的功能。这些功能可以按照优先顺序以表格的形式列出来。这种风险级别的排列考虑了损失发生的可能性以及由于损失带来的影响的严重程度。

风险分析在整个测试活动的各个阶段几乎都可以得以应用，事实上，风险分析的结果直接影响我们在测试中的决策。对项目中风险级别最高的部分要做最多最充分的测试，这是基于风险的测试活动中一个基本的原则，以确保最具破坏性的错误能被尽早尽全发现。风险分析还要列出每一项测试活动中可能出现的影响因素或者破坏性因素，以尽早制定规避措施。

风险分析的前提是风险识别，风险识别是一项贯穿于项目实施全过程的项目风险管理工作。项目风险识别的主要内容包括如下几个方面：

- 识别并确定项目有哪些潜在的风险；
- 识别引起这些风险的主要因素；
- 识别项目风险可能引起的后果。

风险可按照不同的标志进行分类，主要的风险类别大体有六类，如图 8-4 所示，分别为风险发生概率、风险后果严重程度、风险引发原因、风险造成的后果、风险发生对象和风险关联程度，通过这种分类有助于进一步认识项目风险及特性。

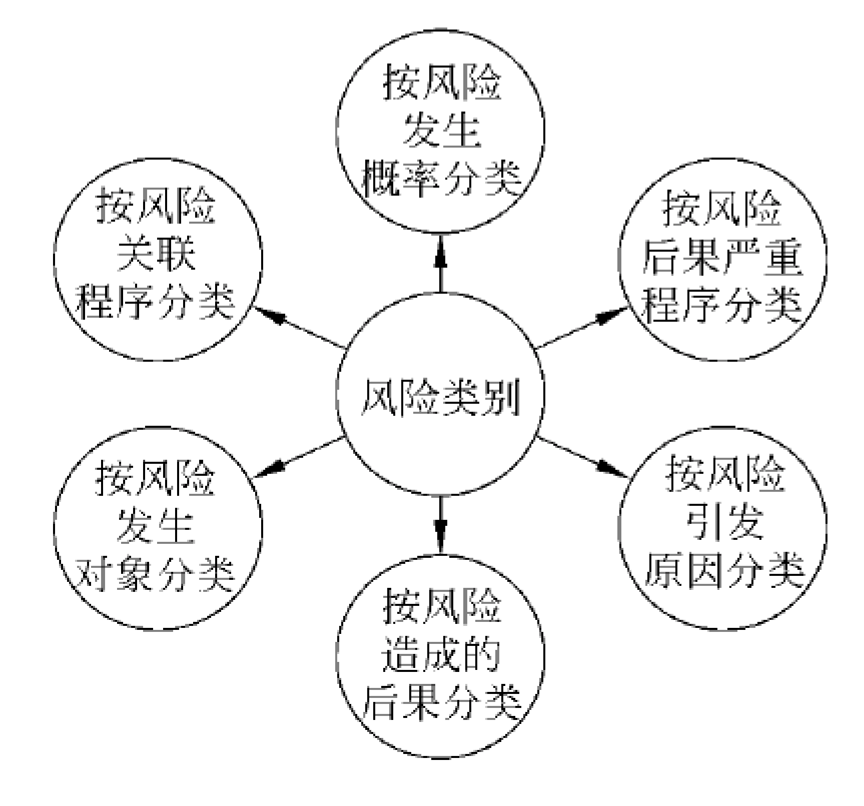


图 8-4 风险类别

项目风险有多种来源，按照风险的来源进行分类具体可以有以下几种。

1. 技术、质量或绩效风险

一般来说，项目中采用新技术或技术创新无疑是提高项目绩效的重要手段，但这样也会带来一些问题，许多新的技术未经证实或并未被充分掌握，则会影响项目的成功。

另外，迫于市场竞争的压力，决策者会盲目提高项目产品性能、质量方面的要求，而罔顾企业自身承受能力也是项目风险的一个来源。

2. 项目管理风险

项目管理风险包括项目过程管理的方方面面，如项目计划的时间、资源分配（包括人员、设备材料）、项目质量管理、项目管理技术（流程、规范、工具等）的采用以及外包商的管

理等。

3. 组织风险

项目决策时所确定的项目范围、时间与费用之间的矛盾是导致组织风险的重要因素。项目范围、时间与费用是项目的三个要素，它们之间相互依赖又相互制约。不合理的匹配往往会导致项目执行的困难，影响项目正常进度。项目资源不足或资源冲突方面的风险同样不容忽视，如人员到岗时间、人员知识与技能等。组织中的文化氛围同样会导致一些风险的产生，如团队合作和人员激励不当导致骨干人员离职等。

4. 外部风险

外部风险顾名思义主要考虑的范围是可能发生在项目组外部的但是又会对项目产生直接影响的一些因素，比如企业主导产品的更替、企业间的合作、并购与重组等。外部风险又可分为外部可预测风险和外部不可预测风险。

影响项目的风险各种各样，要确保项目能够按照预期的工作计划顺利进行，尽可能完整准确地识别项目风险是一个重要的前提条件。一般来说，项目风险识别的方法主要有以下几种：

（1）文档审查。通读项目移交前后的有关资料，从整体到范围细节的层次上进行系统的审查，往往能够发现其中潜在的风险。这既是一个风险排查的过程，同时也有助于测试团队成员尽快熟悉业务。

（2）信息收集技术。

① 头脑风暴法。头脑风暴法简言之就是以圆桌会议的方式在一位主持人的推动下，参加会议的每个人员都发表自己所看到的或认为的风险，由主持人记录后逐一审议、识别、归类。图 8-5 所示的是发生在一次测试项目中的一份头脑风暴成果。

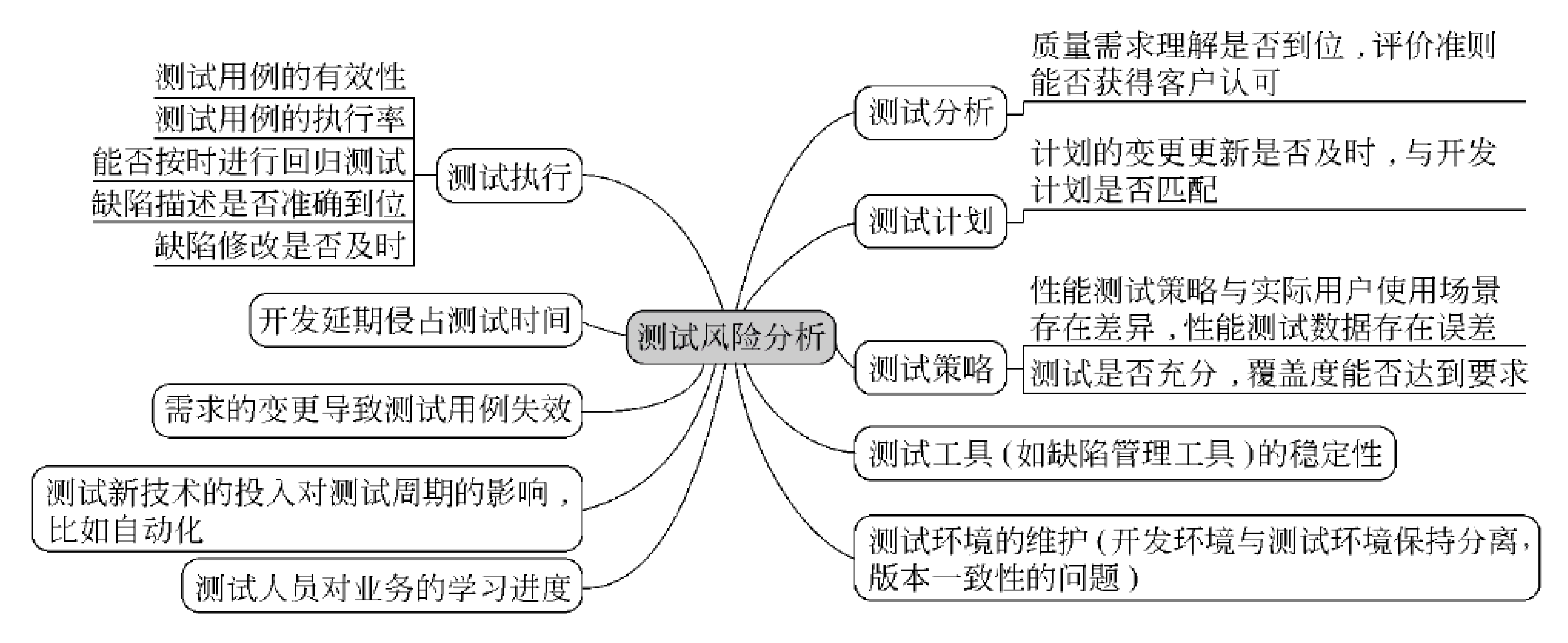


图 8-5　测试风险分析的头脑风暴结果

② 会谈。积极与项目经理及有关干系人进行沟通，了解项目的更多背景有助于发现潜在的风险。

③ SWOT 分析。SWOT 分析常常用于战略规划的过程，用以进行与竞争对手的机会、威胁、优势和劣势分析。应用于风险识别中，可以保证团队从态势分析的每个角度审

议，以扩大所考虑风险的范围。

（3）检查清单。风险识别的一个有效方法是建立检查清单，按照风险类别逐一列出本类别可能存在的风险因素，按照风险内容进行逐一筛查。常见的出现在软件测试中的风险列于表 8-2 中。

表 8-2 常见软件测试风险

风险所属类别	风险因素	说　　明
技术、质量或绩效风险	测试工具的限制； 测试环境与真实环境的差异； …	测试工具本身会有一定的误差； 测试基础数据不足以模拟真实的用户历史数据； …
组织风险	团队成员的稳定性、责任感、工作态度； …	长时间工作失去兴趣，缺乏积极性； 员工请假，造成人力资源紧张； …
	员工对所测业务缺乏了解，上手慢； …	对参与测试的员工培训不足； …
项目管理风险	需求变更风险； …	需求变更导致原有的测试用例失效； …

除了以上介绍的几种风险识别方法之外，另外还有假定分析和图解技术等方法，这些在有关的项目管理资料中基本上均可以查阅到，在此不再一一赘述。其中，关于图解技术，我们重点说一下“原因结果分析图”，这个在日常工作中应该说还是有比较广泛的应用。

“原因结果分析图”也称“鱼骨图”，因为由主要原因、次要原因等构成的分析图看起来很像一个吃剩的鱼骨而得名。如图 8-6 所示，即是某个测试项目从技术方面、组织方面和供应商方面（外部因素）进行分析后得到的原因结果分析图，图中每一个原因都可能是一个项目风险因素。

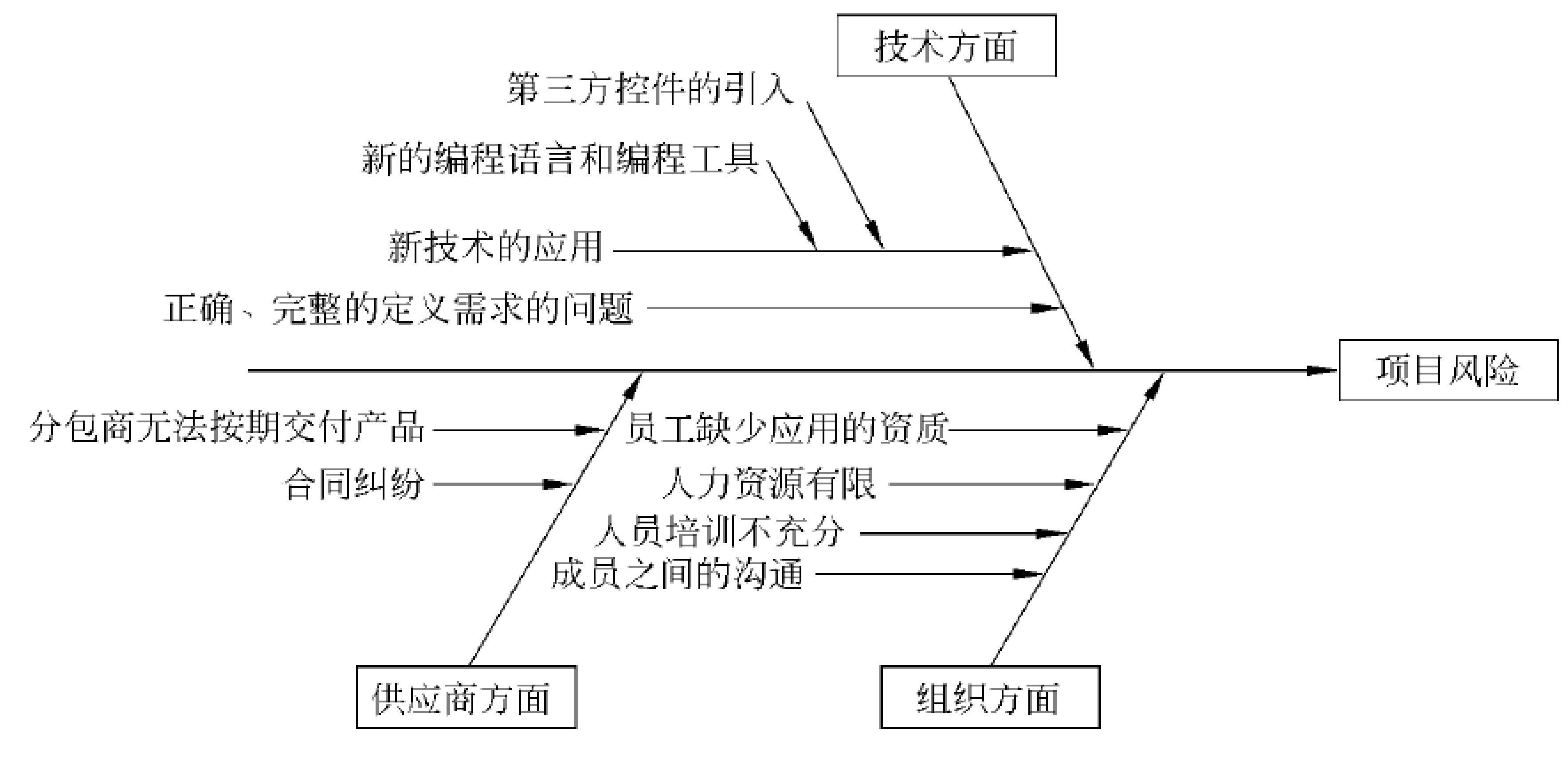


图 8-6 因果图实例

以上描述了风险的分类和识别方法，那么在风险识别之后应该能够得到一张风险清单，最终整理之后形成《风险一览表》(如表 8-3 所示)。通过前面所述内容的学习，这个表中的大部分项目，我们现在已经可以填入，唯一不能确定的就是阴影标注区域，也就是“风险等级”。

表 8-3　风险一览表

编　　号		填表人		填表日期		
项目名称				项目经理		
风险编号	风险名称	风险类别	发生概率	风险后果	风险期望值	风险等级

对于风险应该进行系统化的管理，一般风险处理的步骤是：识别风险、分析风险、规避风险，风险不可能消除，只能有意识地去规避。所谓风险管理是指在一个项目或在组织内的系统的确认、分析/评估、控制，指导克服风险的过程。风险分析表是常用的风险分析方法，如图 8-7 所示。图中按照识别到的风险一旦出现时的危害严重程度分为“无危害或极小危害”、“显著危害”和“实质性危害”三个等级(对应于图 8-6 横向表头，危害大小顺序按照 3、2、1 标识)，按照风险出现的可能性也可以划分为“不太可能”、“可能”和“很可能”三个等级(对应于图 8-6 纵向表头，同样按照可能性大小顺序以 3、2、1 标识)，危害程序和可能性聚焦在一起就形成一个风险分析矩阵，也就是风险分析表。在这个矩阵中，每一行和每一个列的交叉构成一种风险的处理等级，这个等级一般划分为三个，出现可能性最大、危害最大的标识为“1”，依此类推，直至所有的风险都有一个等级对应。

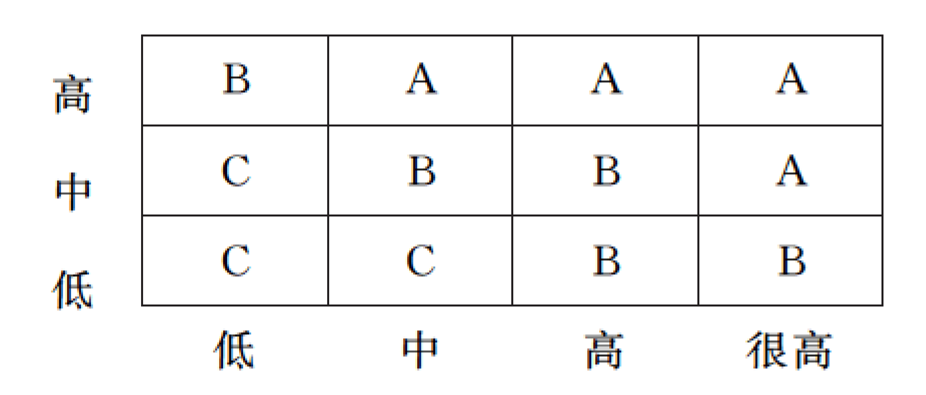
危害程度 / 发生概率	无危害或极小危害	显著危害	实质性危害	
很可能	2	1	1	1
可能	3	2	1	2
不太可能	3	2	2	3
	3	2	1	

图 8-7　风险分析矩阵

那么这个等级是如何来确定的呢？风险的定量分析是根据计算值来区分优先级，具体的计算方法可以表达如下：

$$\text{Risk} = \text{Probability} \times \text{extent of damage}(\text{风险} = \text{可能性} \times \text{危害程度估计})$$

关于这个计算公式的两个运算数都是一个概率性的值，关于这两个值的确认是一个困难。比较同行的做法是专家判定法，即让比较有经验的专家根据以往的项目经验进行估值或者多位专家进行估值然后取均值。在没有同类项目经验的情况下，也可以采用头脑风暴的模式由所有参与者共同估计取平均值来确定。下面这个矩阵(如图 8-8 所示)可视作一个风险处理等级确定的标准参考表，一般情况下处于橘红色区域的风险应首先加以注意，并制定相应的规避措施。



高	B	A	A	A
中	C	B	B	A
低	C	C	B	B
	低	中	高	很高

图 8-8　风险分析等级

在测试风险分析后要检查确认，找出规避措施，尽可能的降低风险带来的对测试项目的冲击，尤其要避免对测试结果产生直接负面影响的风险因素。

常见的测试风险控制的方法简言之可以归结为 6 个字：规避、转嫁、降低。指望排除所有的风险事件是一件不可能完成的任务，但某些具体风险则是可以回避的。正应了那句“大事化小，小事化了”的俗话，大的风险假设是因为任务安排的次序引起的，则可以通过调整原有工作计划来规避，如果是外部因素，比如合作伙伴自身问题，则可以通过分解为更细的包找更多的合作伙伴参与的备份机制来避免。这是讲规避，而有的风险虽然一旦发生可能对我们的结果产生很严重的影响，但是可以通过其他的变通渠道将它转化为其他的较低一级的风险而避免大的损失的出现，这是通过转嫁而实现了风险的规避或者风险等级的降低。

当然，对于一些实在没有好的办法去应对的风险，只要在允许范围内也只能选择接受，否则也就意味着项目的失败。

在对所有风险加以处理之后，得到《风险应对计划》作为项目执行期间的风险处理指导文件(见表 8-4)。

表 8-4　风险应对计划

文档编号		填表人			填表日期	
项目名称					项目经理	
风险编号	风险名称	风险期望值	风险等级	风险战略	应对措施	风险应对负责人

另外，需要注意的是，风险不是一成不变的，它随着项目状态的迁移也在演化，所以对于已识别的风险还要实时的加以监控，必要时要修订《风险应对计划》甚至要重新进行风险识别，这也正是为什么在图 8-3 中有多次风险分析活动参与的原因。事实上，一次软件测试本质上也是一次不断消除软件项目风险的过程，如果你认同软件缺陷本身就是项目的最大风险的话。

【案例项目风险分析实例】

第三方测试全景回放：

针对 HRMIS 的测试项目，由项目负责人召集全体测试组成员和资深的测试工程师共同进行了一次评审。在评审会议上，所有人员根据当前组织的测试技术能力参与讨论了项目的可行性、现存在的一些问题以及可能出现的问题。会后，项目负责人认真整理了这些信息，最后集中了如图 8-9 所示的几个问题。

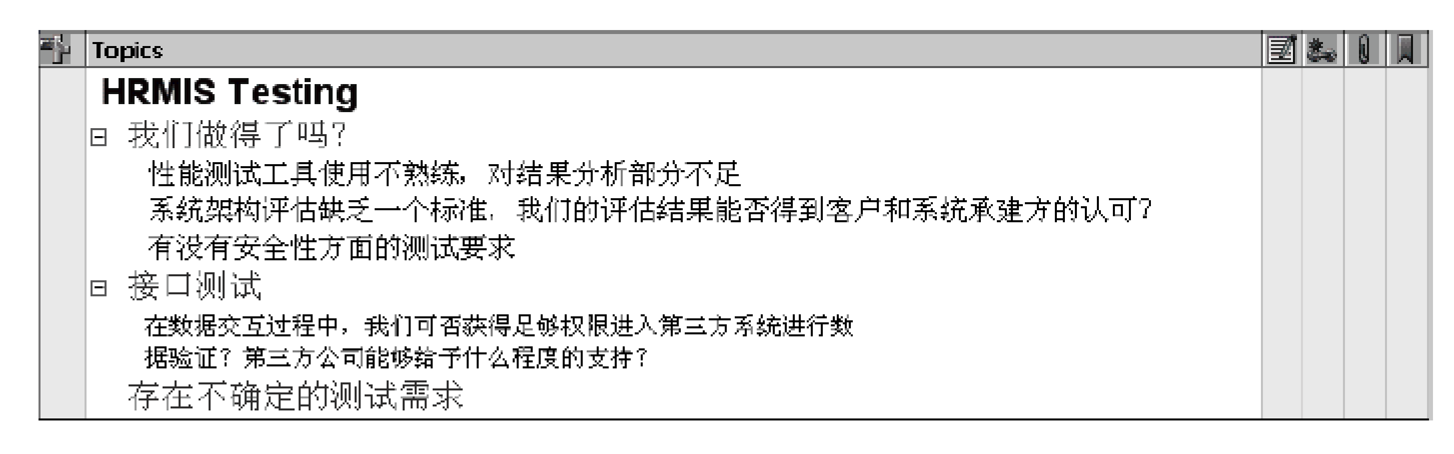


图 8-9　风险分析头脑风暴

上述过程本质上是一次头脑风暴的过程，会议的结果展现出来就是一次很好的风险分析成果，因为正是这些悬而未决的问题构成了影响项目的最大风险。下面按照风险分析的过程重新来梳理一下这些问题。

“我们做得了吗?”这些问题集中反映的是技术能力的问题，属于前面所述的技术风险。“接口测试”这部分主要还是看第三方系统的原承建方或者甲方的配合问题，这个是可以预测的外部风险。“不确定的测试需求”应该是我们尽量避免的，应该在正式签署工作协议前就要力争解决掉，否则就尽量说服客户不纳入测试范围之内。这样，现在就可以给出 HRMIS 的项目分析表了(见表 8-5)。

表 8-5　HRMIS 项目风险分析

编　　号	—	填表人	—	填表日期	—
项目名称	人力资源管理系统(HRMIS)系统测试			项目经理	—
风险编号	风险名称	风险类别	发生概率	产生后果	风险等级
1	性能测试工具掌握程度	技术风险	很可能	实质性危害	
2	系统架构评估标准认可	技术风险	可能	显著危害	
3	第三方系统承建方支持度	外部可预测风险	可能	显著危害	

在上面这个风险评估的基础上，依据风险等级评估公式可以给出各个风险的等级，得到下面这张完整的风险评估表(表 8-6)。

表 8-6　HRMIS 风险评估

编　　号	—	填表人	—	填表日期	—
项目名称	人力资源管理系统(HRMIS)系统测试			项目经理	—
风险编号	风险名称	风险类别	发生概率	产生后果	风险等级
1	性能测试工具掌握程度	技术风险	很可能	实质性危害	A
2	系统架构评估标准认可	技术风险	可能	显著危害	B
3	第三方系统承建方支持度	外部可预测风险	可能	显著危害	B

针对不同风险的特点,需要制定一些有效措施加以规避(或者转嫁),本例中可以制定以下措施:

① 风险 1:性能测试工具掌握程度。

可以采取的措施:

- 安排外部培训;
- 租借有经验的人员加入项目组。

② 风险 2:系统架构评估标准。

可以采取的措施:

- 客户、系统承建方和测试实施单位共同拟定评估标准;
- 听取有经验的系统架构师的建议。

③ 第三方系统承建方支持度。

可以采取的措施:

- 列出支持项,递交给客户,争取主动权;
- 争取客户的系统管理人员作为备份。

8.3.3　测试范围的识别

测试范围的界定,简单地说就是测试活动需要覆盖的范围,通俗地说就是要对哪些东西的哪些方面进行测试。测试范围常常意味着测试目标的界定,在某些参考资料上有时可能会发现,测试范围和测试目标被放在一起。

测试范围本质上是一次在有限测试资源的情况下进行测试的权衡和取舍的过程。对于这个过程,可以稍加总结,以"先功能再其他,先明确后模糊,最后资源修正"这样一句话加以概括。

1. 先功能再其他

测试对象所实现的功能永远是各种测试的基础,功能也是计量的天然单位,通过对软件功能的统计,可以很快勾勒出软件测试的范围,即哪些功能需要测试、哪些功能在某个阶段暂时不进行测试。对于非功能测试,可以从用户实际使用场景去考虑,从性能测试、安装测试、稳定性测试等方面去选择进行。

2. 先明确后模糊

测试范围开始时可能是不明确的,在确定测试范围时尽量先易后难,先把比较明确的

确定下来，然后随着软件项目的进行再逐渐修订和增补测试范围。事实上，在时间约束，产品质量约束的情况下，在软件测试过程中唯一能够调整的就是范围。在实际的工作中，总是在自觉不自觉地调整软件测试的范围，如在时间紧张的情况下，通常要优先保障核心功能的测试。

3. 最后资源修正

影响测试范围的两个主要因素是时间和人力，时间充足、人力充足的情况下，可以适度地扩大测试范围以达到更多的测试覆盖，保障产品质量；反之，可以提请项目经理或有关负责人缩小测试范围，将有限的测试资源投放到最急需的部分（如图 8-10 所示）。

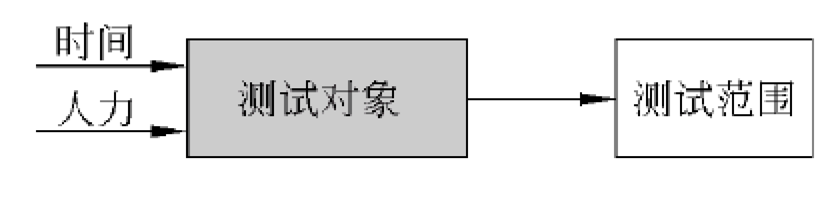


图 8-10　资源修正示意

确定 HRMIS 的测试范围

结合 HRMIS 的需求说明书和系统设计说明书（图 8-11 是从业务需求说明书中抽取的框架图），可以分析出 HRMIS 的系统功能，见表 8-7。

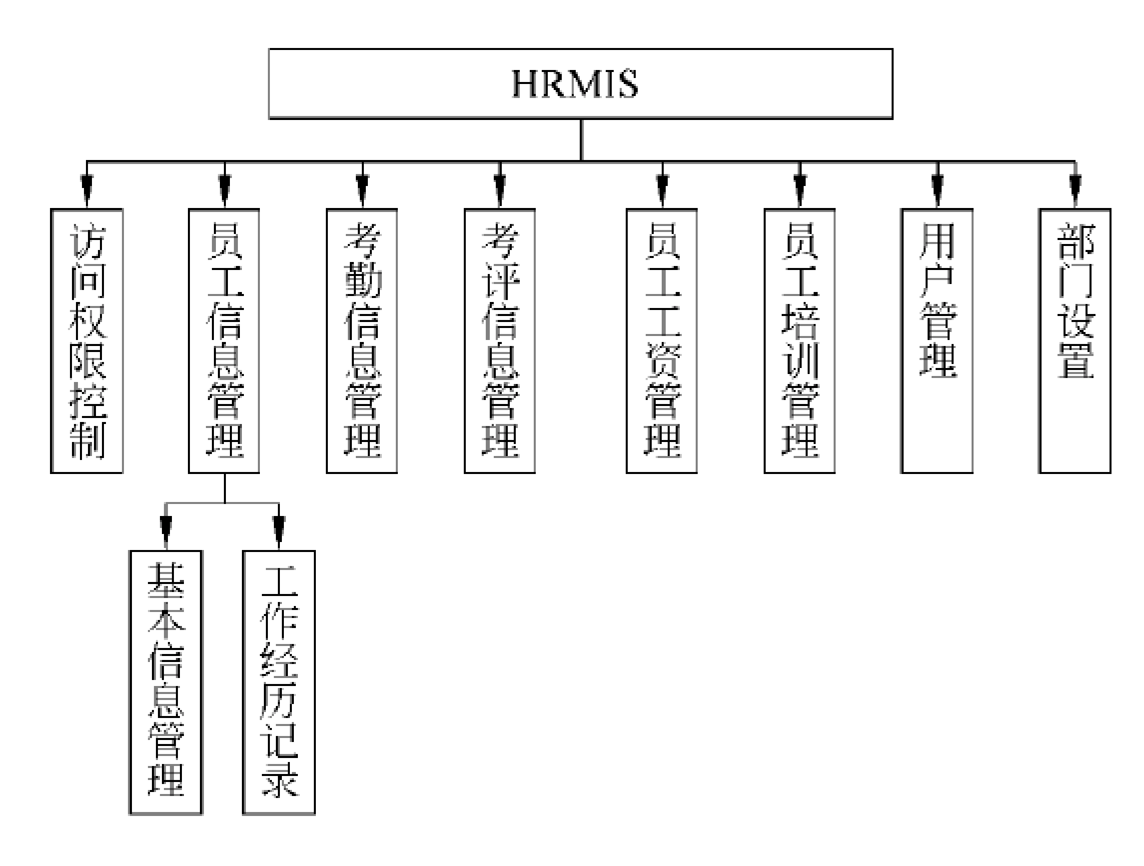


图 8-11　HRMIS 系统框架

表 8-7　HRMIS 系统功能清单

一级模块	二级模块	三级模块
员工信息管理	新增员工信息	
	修改员工信息	
	删除员工信息	
	部门调动	
	合同管理	合同签订
		合同修改
		合同删除

续表

一级模块	二级模块	三级模块
员工信息管理	工作履历	添加工作履历
		修改工作履历
		删除工作履历
	参加培训记录	
	培训总结	
	培训考核	
	信息变更历史记录	
培训信息管理	培训信息添加、修改和删除	
	培训机构添加、修改和删除	
	培训信息的培训机构变更	
系统管理	系统用户添加、修改和删除	
	系统用户权限设置	

这个就是功能测试范围，另外在 HRMIS 的需求说明书中还有一部分是性能方面的需求，因此针对 HRMIS 的测试除了功能测试之外，还应包括一部分性能测试工作。性能测试的测试范围如下：

- 系统登录；
- 更新处理；
- 数据转换和传送。

据此，从质量特性角度可以确定测试范围（见表 8-8），有关测试需求分析的过程和方法请参阅第 9 章内容。

8.3.4 制定测试策略

如何确定测试策略呢？先来重温一下有关测试的几个方面，因为大部分在第二篇中已有论述，所以在此只是简单把它们列出，之所以如此，那是因为软件测试的这几个方面事实上总是或明或暗地在影响制定测试策略。

① 测试是昂贵的（需要较多的资源投入），因此尽量只将基于风险的一些必要的测试活动纳入并且必须纳入进来。

② 测试本身没有非常明确的终止点。

③ 测试结果由谁接收并认可，谁为这些结果支付费用。

④ 穷尽测试是不可能的。

⑤ 构建适当的测试集。

测试策略的目标是尽可能早的发现存在于软件中的缺陷，以尽可能小的代价去纠正软件中的缺陷。在测试策略设计过程中，任务的约束条件（包括风险、资源、时间限制以及

表 8-8　测试范围

软件质量特性	子特性	是否测试	软件质量特性	子特性	是否测试
功能性	适合性	Y	效率	时间特性	Y
	准确性	Y		资源特性	Y
	互操作性	Y		依从性	N
	安全保密性	Y	维护性	易分析性	N
	依从性	Y		易改变性	N
可靠性	成熟性	Y		稳定性	N
	容错性	Y		易测试性	Y
	易恢复性	Y		依从性	N
	依从性	N	可移植性	适应性	Y
易用性	易理解性	Y		易安装性	Y
	易学性	Y		共存性	Y
	易操作性	Y		易替换性	Y
	吸引性	Y		依从性	Y
	依从性	Y			

预算的限制等)是必须要加以考虑的。测试策略用以描述哪些需求和风险被什么样的测试覆盖。对于每种测试,都应提供测试说明,并解释其实施的原因。制定测试策略时所考虑的主要事项有:将要使用的技术以及判断测试何时完成的标准。如图 8-12 所示,是软件测试的基本信息流,从软件测试的信息流来看,测试包含输入和输出,以及对这些信息进行处理的过程。作为表征测试的方法和工具的测试策略来说,也可以从输入、输出和过程三个方面去阐述。

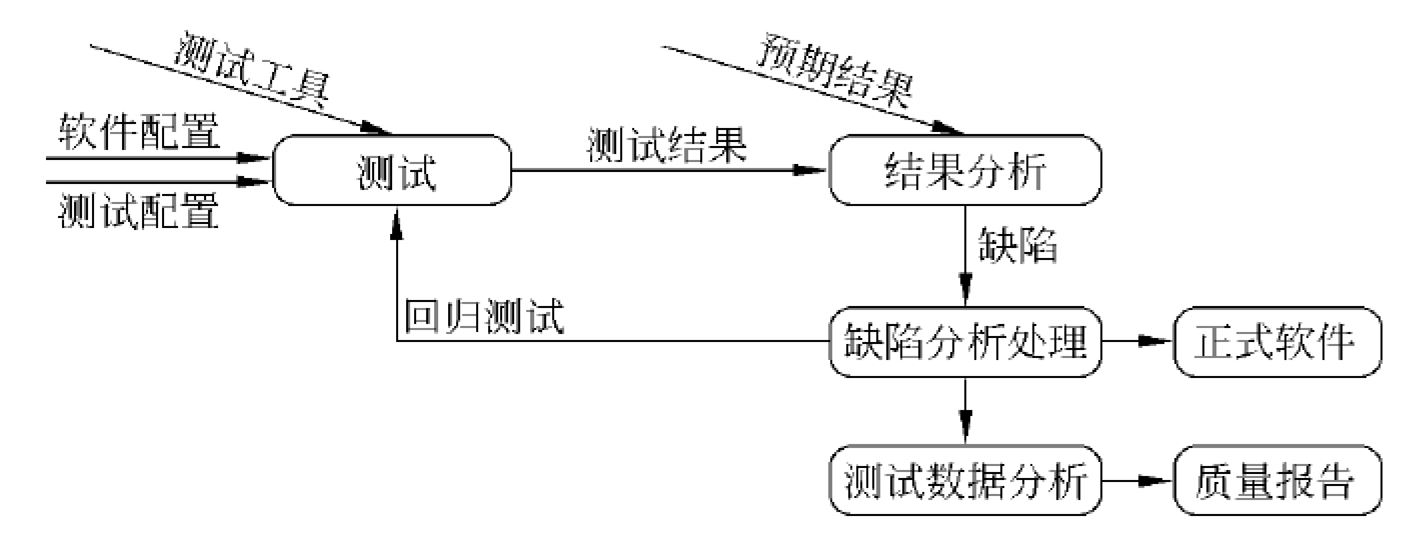


图 8-12　软件测试信息流示意

1. **输入**

- 软件测试环境的搭建要求,所采用的支撑软件、硬件规格,与软件实际运行环境的差异描述;
- 选用什么样的测试工具,测试工具提供哪方面的数据;测试工具的来源,公正性

说明；
- 测试方法说明；
- 测试准入、准出条件。

2. 输出
- 测试主要产出物，如测试计划、测试用例、缺陷报告等；
- 风险应对计划。

3. 过程

(1) 确定测试需求。测试需求是测试策略产生的原因，什么样的测试需求导致什么样的测试策略。关于测试需求部分可以参考第 9 章内容，在此不再赘述。

(2) 评估风险并确定优先级。有效的测试需要在测试活动中科学地权衡资源约束和各类影响软件测试的风险因素。为此，应该确定测试工作的优先级，对于最重要、最有意义或风险最高的用例或构件予以优先测试。而风险分析是确定测试优先级的有效手段，关于风险分析可参阅本章有关内容。

(3) 确定测试策略。测试策略应该包括测试目标、测试范围、测试类型（或层次）、实施测试的技术、测试的开始完成标准以及对测试策略所述的测试工作存在影响的特殊事项等内容（表 8-9 是一个有关功能测试的模板示例）。

表 8-9　功能测试策略示例

测试目标	确保测试的功能正常，包括导航、数据输入、处理和检索等功能
测试范围	可以是测试需求
测试类型	功能测试
技术描述	使用等价类、边界值、因果图等用例设计方法设计测试用例。测试用例至少要包含一个有效的和一个无效的数据，以核实以下内容： • 在使用有效数据时得到预期的结果 • 在使用无效数据时显示相应的错误消息或警告消息 • 各业务规则都得到了正确的应用
开始标准	满足了规定的预置条件，指定功能已通过集成测试，测试用例设计、评审通过
完成标准	测试用例执行率 100%；测试用例通过率 100%
测试重点和优先级	略
需考虑的特殊事项	确定或说明那些将对功能测试的实施和执行造成影响的事项或因素（内部的或外部的）

一个有效的测试策略制定方法可以用图 8-13 来表示。在这个方法中一共有 7 个步骤，其中通过步骤①～步骤③，可以画出第一道分割线上右边的表格，这个表格是一个示例，假设要从准确性、适合性、成熟度、易理解性和时间特性这 5 个方面对目标软件进行测试，这 5 个特性的相对重要性分别设定为 30、20、15、20、15（以上数值均为百分数）。对于目标系统我们要进行单元测试、集成测试和系统测试三个层次的测试活动，那么根据质量特性的相对重要性把这 5 个质量特性分散到这三个层次的测试活动中，而无须每一个层

次的测试都进行这 5 个特性的测试，从而节省宝贵的测试时间同时又能保证测试覆盖要求。

步骤④～步骤⑥构成制定测试策略的第二个大步骤，如图 8-13 所示，对第一步的成果继续深化分析，选择单元测试这个测试活动，这个单元可以进一步细分为 UnitA、UnitB 和 UnitC 三个子单元。根据第一步的成果，单元测试需要对准确性、适合性和成熟度这三个质量属性进行测试，所以在这一个步骤中，表格的列标题设置为准确性、适合性和成熟度，行标题设置为 UnitA、UnitB 和 UnitC。通过行列交叉点标记要进行的测试，这个工作可以结合风险分析的结果和特别指明的测试要求。

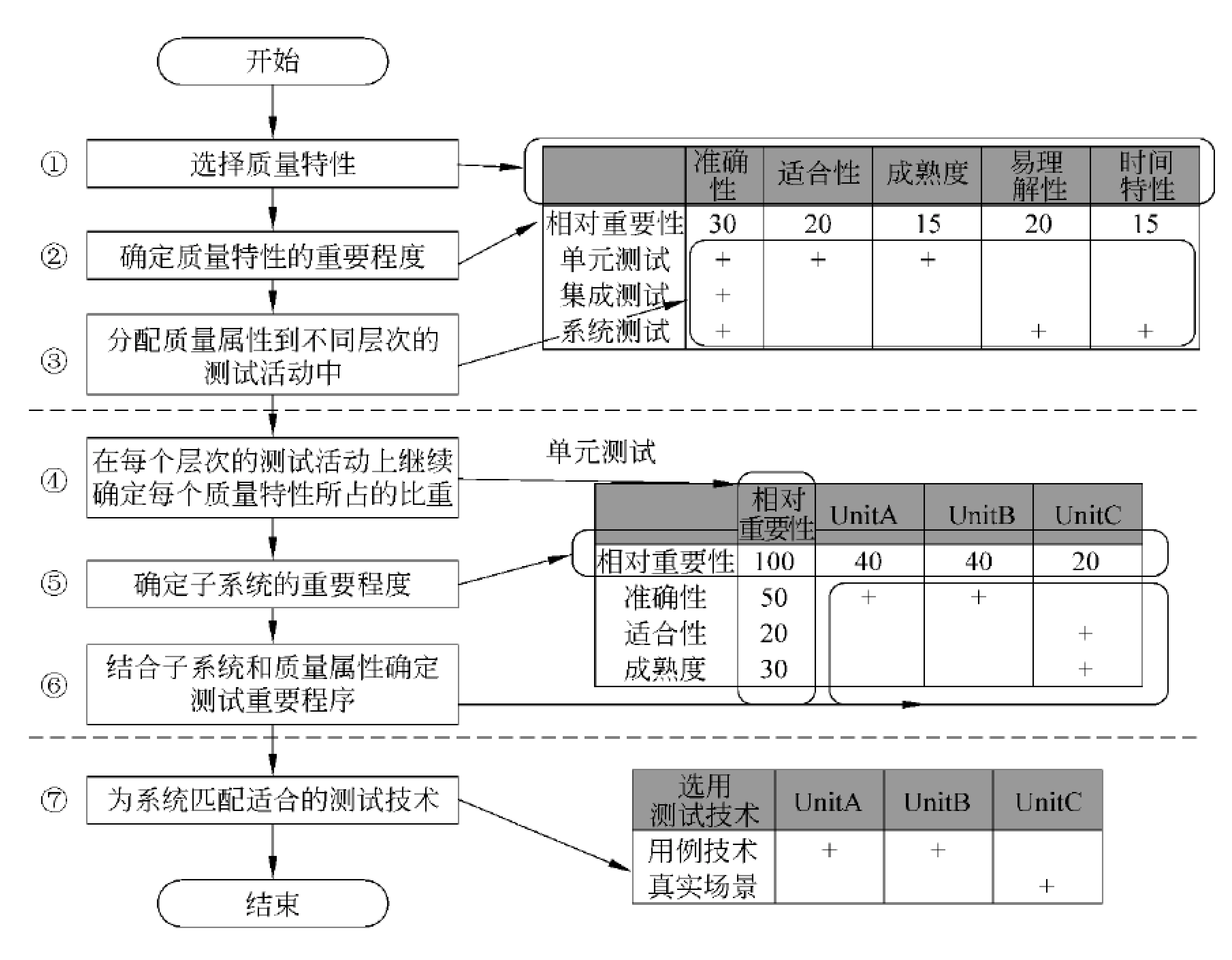


图 8-13 七步策略法示意图

步骤⑦是对选定的测试指定适合的测试技术，例如本例测试技术主要有两类，一类是用例技术，另一类是真实场景技术(即放到用户生产系统中去实地测试)。为了方便记忆，不妨为这个测试策略制定的方法起名字，叫做七步策略法。

测试策略制定的七步策略法实际上综合应用了两种软件测试策略制定方法，即基于测试技术的测试策略和基于测试方案的测试策略，下面分述如下：

(1) 基于测试技术的测试策略。著名软件测试专家 Myers 提出了使用各种测试方法的综合策略：

- 在任何情况下都必须使用边界值分析方法，经验表明用这种方法设计出测试用例发现程序错误的能力最强。
- 必要时用等价类划分方法补充一些测试用例。
- 用错误推测法再追加一些测试用例。

- 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例。
- 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法。

（2）基于测试方案的测试策略。

- 根据程序的重要性和一旦发生故障将造成的损失来确定它的测试等级和测试重点；
- 认真研究，使用尽可能少的测试用例发现尽可能多的程序错误，测试也是过犹不及。

测试策略是软件测试中一个相对较为抽象的概念，有很多人往往与测试方法混为一谈。通过本节的讲述应该知道测试策略是一个系统的测试应对方略，比测试方法有更多的内涵和外延。当然也无须望而生畏，测试策略本质上是一个测试方法的产生过程，所以“怎么想就怎么做，怎么做就怎么说”，这就是测试策略。

HRMIS 的测试策略如下，对于关键模块进行单元测试（见表 8-10），核心功能模块集成要进行集成测试（见表 8-11），系统组装完成后进行系统测试（见表 8-12）。

表 8-10　单元测试策略

测试目标	确保所测试的单元功能正常，没有冗余语句
测试范围	模块：所有中间处理逻辑单元质量特性：功能性（适合性、准确性）、维护性
测试类型	单元测试
技术描述	使用 CppUnit 开发、执行测试用例，检验单元提供的方法 使用 Logiscope 对单元的过程调用关系、函数控制流进行分析使用 Rational TestRealTime 对单元进行语句覆盖分析
开始标准	模块代码编写完成，调试无误
完成标准	所预定的测试覆盖（语句覆盖、路径覆盖等）达到目标值 所预定的测试用例执行通过
测试重点和优先级	关键模块的处理逻辑单元优先
需考虑的特殊事项	测试工具对编程语言的支持度 驱动模块和桩模块的设计和编制

表 8-11　集成测试策略

测试目标	确保集成后的系统模块的功能正常，参与集成的单元相互数据传递无误
测试类型	集成测试
测试范围	UI（User Interface）与中间处理逻辑、数据库的集成
技术描述	（1）选定集成测试方案，如自顶向下、自底向上等 （2）执行集成测试用例，利用有效的和无效的数据来执行各个用例、用例流或功能，以核实以下内容： • 在使用有效数据时得到预期的结果 • 在使用无效数据时显示相应的错误消息或警告消息 • 各业务规则都得到了正确的应用

续表

开始标准	单元测试通过
完成标准	集成测试用例执行通过
测试重点和优先级	业务优先：员工信息管理、培训信息管理 共享度优先：优先测试公用程度高的部分
需考虑的特殊事项	开发进度可能会影响集成测试方案

表 8-12　系统测试策略

测试目标	确保测试的功能正常，其中包括导航，数据输入，处理和检索等功能
测试类型	系统测试(功能测试)
测试范围	功能性、效率、可维护性、可移植性等
技术	使用等价类、边界值、因果图等用例设计方法设计测试用例 测试用例至少要包含一个有效的和一个无效的数据，以核实以下内容： • 在使用有效数据时得到预期的结果 • 在使用无效数据时显示相应的错误消息或警告消息 • 各业务规则都得到了正确的应用 确定关键业务，使用 LoadRunner 开发自动化性能测试脚本，仿真 HRMIS 的实际应用场景，通过压力测试评估系统各项性能指标
开始标准	指定功能的集成测试通过
完成标准	系统测试用例达到规定的通过率 缺陷得到有效处理，遗留缺陷的影响做了充分估计
测试重点和优先级	功能性和效率测试
需考虑的特殊事项	效率测试的环境问题

8.3.5　测试资源评估

在以 H 模型为指导的测试组织中，软件测试常常是作为一个独立的测试项目(第一方/第二方可能是隐式的，第三方是显式的)来组织管理。其间很多可以借鉴软件开发项目的项目管理模式和思想。

测试资源的准确评估是测试计划阶段一个重要的内容，直接影响最终测试目标的达成。综观软件测试，作为测试资源的可以分为以下几类：人力资源、测试时间(起止时间)、测试工具、测试设备、项目资金预算(第三方测试)和外部协调资源(如图 8-14 所示)。

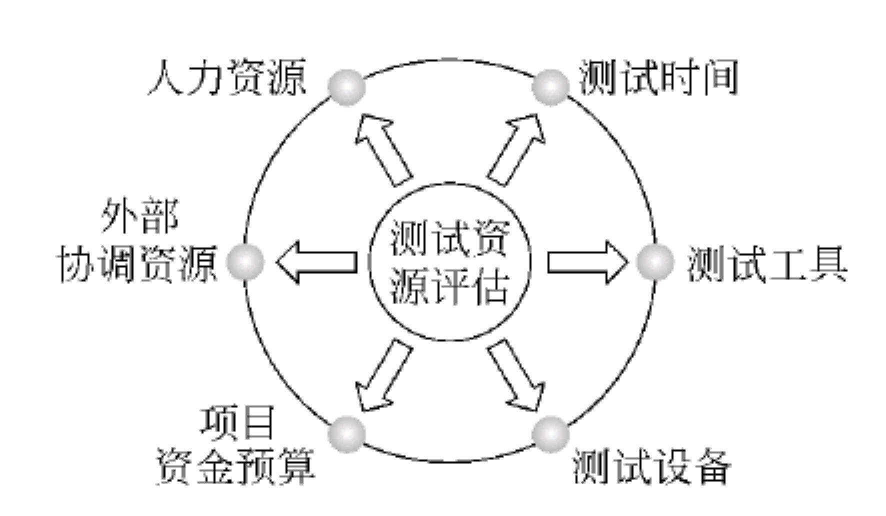


图 8-14　测试资源评估

1. 人力资源

人力资源是影响测试进度的直接因素之一，人员充足可以保证用较短的测试周期完

成更多的测试覆盖。人力资源的主要考虑因素是数量和人员的业务背景、测试技能等。在一个相对成熟的测试团队中，一般由以下几类人员组成(如表 8-13 所示)。

表 8-13　测试团队组成示例

工作角色	具 体 职 责
测试项目负责人	管理监督测试项目，提供技术指导，获取适当的资源，制定基线，技术协调，负责项目的安全保密和质量管理
测试分析员	确定测试计划、测试内容、测试方法、测试数据生成方法、测试(软、硬件)环境、测试工具，评价测试工作的有效性
测试设计员	设计测试用例，确定测试用例的优先级，建立测试环境
测试程序员	编写测试辅助软件
测试员	执行测试、记录测试结果
测试系统管理员	对测试环境和资产进行管理和维护
配置管理员	设置、管理和维护测试配置管理数据库

注：一个人可承担多个角色的工作，一个角色可由多人承担。

2. 测试时间

测试时间是影响测试充分性的直接因素之一。事实上大部分的测试项目测试时间都是十分紧张的，而我们已经知道，软件测试做到穷尽测试是不可能的，因此必须要对测试时间加以考量，把时间合理分配到被测试软件最重要、最有意义的部分去使用。

3. 测试工具

目前，用以支撑软件测试的工具软件十分丰富，这个在本书第五篇中将有详细的说明。针对具体的测试任务选择最经济有效的工具无疑是保证软件测试有效性的一个必要条件。

4. 测试设备

测试设备可以分为测试辅助设备和测试环境搭建设备。前者主要是和被测试系统配合形成相互的信息数据支持，比如在测试有关移动彩信业务时，可能会选择几种品牌的手机在用户真实使用环境下进行最后的联合测试。而后者是进行已定义的各类测试的基础平台(Test Bed)，包含基本硬件(比如客户机、服务器、打印机、传真机等)和第三方服务软件(比如操作系统、应用服务器软件等)。要对测试设备评估，一般是从硬件的配置、软件的版本上考虑。例如，我们的测试策略中要求对兼容性进行测试，则可能需要构建多种操作系统环境，安装多种浏览器去访问目标 Web 服务器，以检验其对浏览器的适应能力。

5. 项目资金预算

软件测试的费用支出主要是用于设备、工具的购买，人员的培训或者外协人员的租用等。当然，对于需要驻外的项目还有差旅上的支出。

6. 外部协调资源

这个主要决定项目需不需要外部人员的参与或者对本团队或组织进行能力评估，如果对某项测试需求暂时无法满足，则与用户沟通之后可以寻求分包加以解决。

关于如何进行资源评估，这里主要介绍一下简单易用的类比估算法。这是一种在项目成本估算精确度要求不是很高的情况下使用的项目成本估算方法。这种方法也被叫做自上而下法，是一种通过比照已完成的类似项目的实际成本，去估算出新项目成本的方法。这种方法的缺点是专家依赖，即要求有足够的历史项目经验和数据。

HRMIS 本质上是一个信息管理系统，通过类似项目的数据推演，可以给出 HRMIS 的测试资源预算。

(1) 人力资源。HRMIS 的整体测试工作按阶段可以分为单元测试、集成测试和系统测试三个主要的阶段，其中单元和集成测试计划由开发人员自行完成，故这部分工作基本上不存在人力资源支出。考虑到系统测试工作量和项目周期，系统测试工作共需 2 或 3 名测试人员来完成，另设一名测试负责人，构成和组织框架如图 8-15 所示。

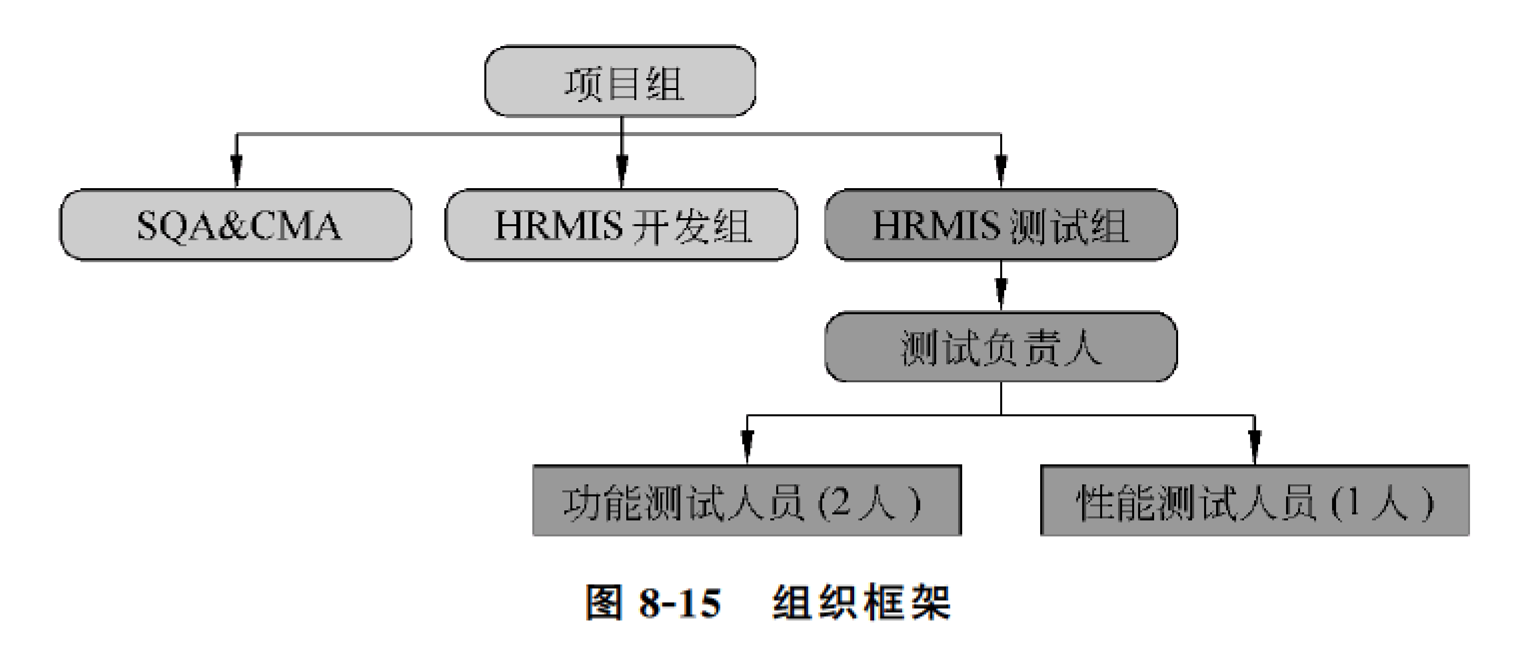


图 8-15　组织框架

(2) 测试时间见表 8-14。

表 8-14　测试时间

测试类型	起始时间	结束时间
单元测试	20××年 12 月 10 日	20××年 12 月 16 日
集成测试	20××年 12 月 18 日	20××年 12 月 29 日
系统测试	20××年 12 月 29 日	20××年 1 月 19 日

(3) 测试设备和工具。根据 HRMIS 的结构，暂时为其配置 3 台测试机，其中 1 台作为数据库服务器使用。测试工具使用 SVN 配置管理工具、TestDirector 8.0 和 LoadRunner 9.1，其中 SVN 可以与项目组共享使用，TestDirector 属于公司的公共平台，不需要另行配置。

8.3.6 计划任务

计划任务是测试计划的核心内容，计划安排合理不合理直接影响测试的进度和测试的效果。综合来看，对计划任务安排有影响的因素有以下几个，为了清晰起见，这里以一张因果图来表示(如图 8-16 所示)。

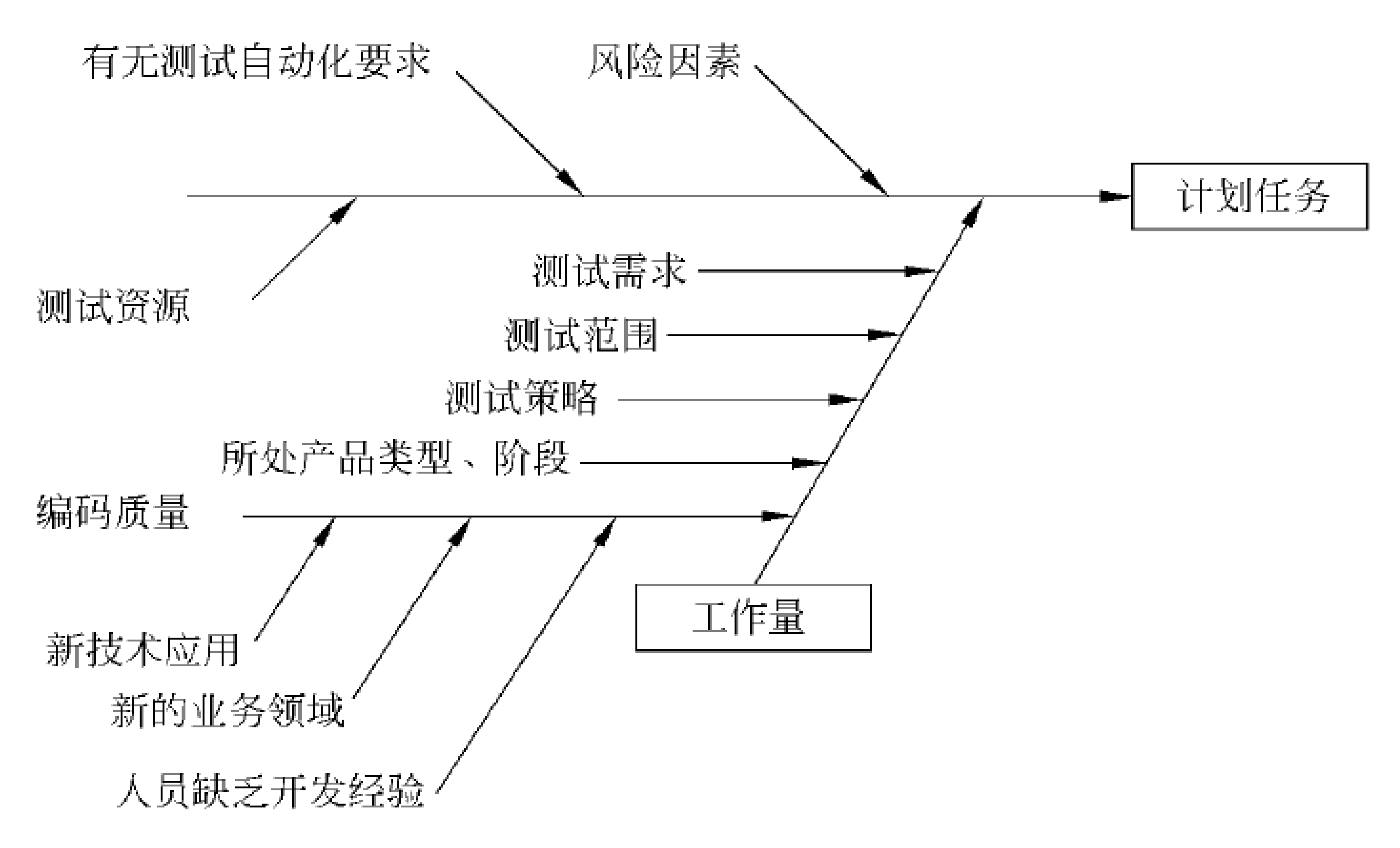


图 8-16　计划影响因素分析

从这张因果图上可以看到，前述的测试需求、测试范围、测试策略、测试资源和风险因素等都是影响计划任务安排的因素，也就是说它们之间都是有着很紧密的内在联系的，厚此薄彼写不出高质量的测试计划。

对于这些因素进一步分析，可以把它们按照正面因素和负面因素来加以区分认识(见表 8-15)。

表 8-15　影响分析

描　　述	影　响	解　　释
风险因素	↓	风险因素决定了计划任务的执行优先级，有时会扰乱原有的工作计划
有无测试自动化要求	↑↓	现在对于测试自动化业界基本有了一个比较理性的认识，它对于测试是个双刃剑，无法忽视为了实现自动化而成倍的资源投入
测试资源	↑	测试资源永远是个积极因素，资源愈充足，测试愈充分
测试需求	↓	测试需求越多，测试的广度和深度都会越大，工作量也在增大
测试范围	↓	测试范围越大，测试的广度和深度都会越大，工作量也在增大
测试策略	↑	测试策略是方法范畴，测试策略越合理越能取得事半功倍的效果
所处产品类型、阶段	↓	稳定的产品测试工作可以复用很多东西，而新产品则需要更多的重新做起
新技术应用	↓	应用新技术往往会引入更多的不稳定性
新的业务领域	↓	新的业务领域，产品往往会有很多不确定性
人员有无开发经验	↑	有经验的开发人员其良好的编码习惯无论对于开发还是测试都有着很好的促进作用

经验表明，对于任务的认识越清晰则估算的工作量越趋近于合理。因此，任务分解成了一个行之有效的任务分析法。其目的是把大一些的任务分解为粒度更小的子任务，子任务当然也可以继续分解为更小的子任务。通过这种分解，把内容繁多的任务变成任务

单一明确的子任务集合。通过对子任务的工作量估算，从而达到对总体任务的工作量估算。这种任务分解有一个专业的名字叫做工作分解结构(Work Breakdown Structure, WBS)。下面将着重介绍一下 WBS 的概念和使用方法。

WBS 广泛的应用于项目管理中，它是以可交付成果为导向对项目要素进行的分组，它归纳和定义了项目的整个工作范围每下降一层代表对项目工作的更详细定义。WBS 总是处于计划过程的中心，是制定进度计划、资源需求、成本预算、风险管理计划和采购计划等的重要基础。WBS 同时也是控制项目变更的重要基础。借鉴于项目管理体系的 WBS 在软件测试领域的主要用途同样可以归结为以下四个方面：

- WBS 帮助测试经理和团队成员有效地确定和管理测试工作。
- WBS 是一个清晰表示各测试工作之间的相互联系的结构设计工具。
- WBS 是一个展现测试工作全貌，详细说明为完成测试工作所必须完成的各项工作的计划工具。
- WBS 定义了里程碑事件，可以向高级管理层和客户报告测试进度。

WBS 可以采用多种方式进行分解，常见的分解方式有按产品的物理结构分解、按产品或项目的功能结构分解、按照实施过程分解、按照项目的地域分布分解、按照项目的各个目标分解、按部门或者职能分解等。在软件测试活动中多以测试的阶段划分进行分解，即“按照项目的各个目标分解”。WBS 的分解不是随意的，在进行 WBS 时应牢记以下原则：

- 保持 WBS 项唯一。某项任务应该在 WBS 中的一个地方且只应该在 WBS 中的一个地方出现。
- WBS 中某项任务的内容是其下所有 WBS 项的总和。
- 保持负责人唯一。一个 WBS 项只能由一个人负责，其他人是参与者。
- 应该让项目团队成员积极参与创建 WBS，以确保 WBS 的一致性。成员在创建 WBS 的过程中也是对项目的深入了解的过程。
- WBS 必须在根据产品需求说明书正常维护项目工作内容的同时，也能适应频繁的变更需求。

创建 WBS 的方法有很多种，最常见的是类比法、自上而下/自下而上法等。类比法适用于有同类项目的 WBS 数据的情景。如某大型 IT 公司在某个省份成功实施了省级集中烟草数据信息管理系统，现将向周边几个省份进行推广应用。则其他推广省份的项目负责人除了可以进行业务、产品复制以外，项目的运作也是可以复制的，尤其是 WBS 项因为产品类同所以也具备充分的复制条件。项目经理只需根据用户的个性化需求对 WBS 项稍加调整即可完成新的项目的 WBS 工作，而且可以相信这种来源于实践的 WBS 具备十足的准确性和可执行性；自上而下法也是一种很常见的 WBS 创建方法，这种方法在不能选择类比法时经常使用。其表现形式即是从项目的顶层任务开始，逐级分解为更好度量的单一任务。自上而下法往往要经过几次反复或者说迭代，以综合可能的评审意见，最终形成正式的 WBS。自上而下法还有一层意思是说这种 WBS 的创建过程是从项目管理者开始的，然后分发给各岗位人员；自下而上法是 WBS 里的头脑风暴，它是从项目团队成员发起，每个人都为此贡献，发表并记录自己想到的可能存在的任务，然后再统

一评审归纳整合。

通常,可以把创建 WBS 的过程描述为以下 6 个步骤:

① 获得产品需求说明书或测试任务说明书。

② 召集有关人员集体讨论主要的测试工作,确定工作分解的方式。

③ 分解项目,画出 WBS 的层次结构图(树状图或者行首缩进的表格)。

④ 将主要项目可交付成果细分为更小的、易于管理的组或工作包。工作包必须详细到可以进行成本和工期的估算、安排进度、做出预算、分配负责人员或组织单位。

⑤ 验证上述分解的正确性。正确性与否,可以通过加和方式进行,即将所有分解得到的子任务合在一起,看是否已经覆盖了所有的产品需求或者任务要求。

⑥ 根据实际项目进展或关联活动进展适时调整 WBS,执行顺序或者完成工期。

已经知道 WBS 是工作任务的分解,WBS 的过程中较少考虑任务间的关系和时间成本。因此说 WBS 只是完成了计划任务工作的基础工作,要制定出一份优秀的计划任务,还需要将任务间的关系(融合了图 8-16 的其他几项因素,如策略、风险等)和时间成本考虑进去。这就是任务排序和任务工期预算。

任务排序是指识别 WBS 中各项任务的相互关联与依赖关系,并据此对项目各项任务的先后顺序的安排和确定工作。要完成这项工作,可以从分析各个任务的输入、输出考虑,以厘清任务间的调用关系、外部依存关系、任务里程碑等。

任务工期估算是根据 WBS 中定义的任务清单估计完成这些任务所需的工期。工期通常以小时或天表示,但大型项目也可能用周或者月作为表示工期的单位。工作量的大小是任务工期预算的主要考虑因素,在图 8-16 中已经对影响工作量的因素做了详细分析,在此不再赘述。那么如何来确定工期呢?本书主要介绍一下系统分析法。

系统分析方法是通过计算出所有任务的最早、最晚开始和结束日期,并且考虑多种因素的影响,编制项目工期计划的方法。其中,关键路径法(CPM)和计划评审技术(PERT)都是属于系统分析法的范畴。

1)关键路径法(CPM)

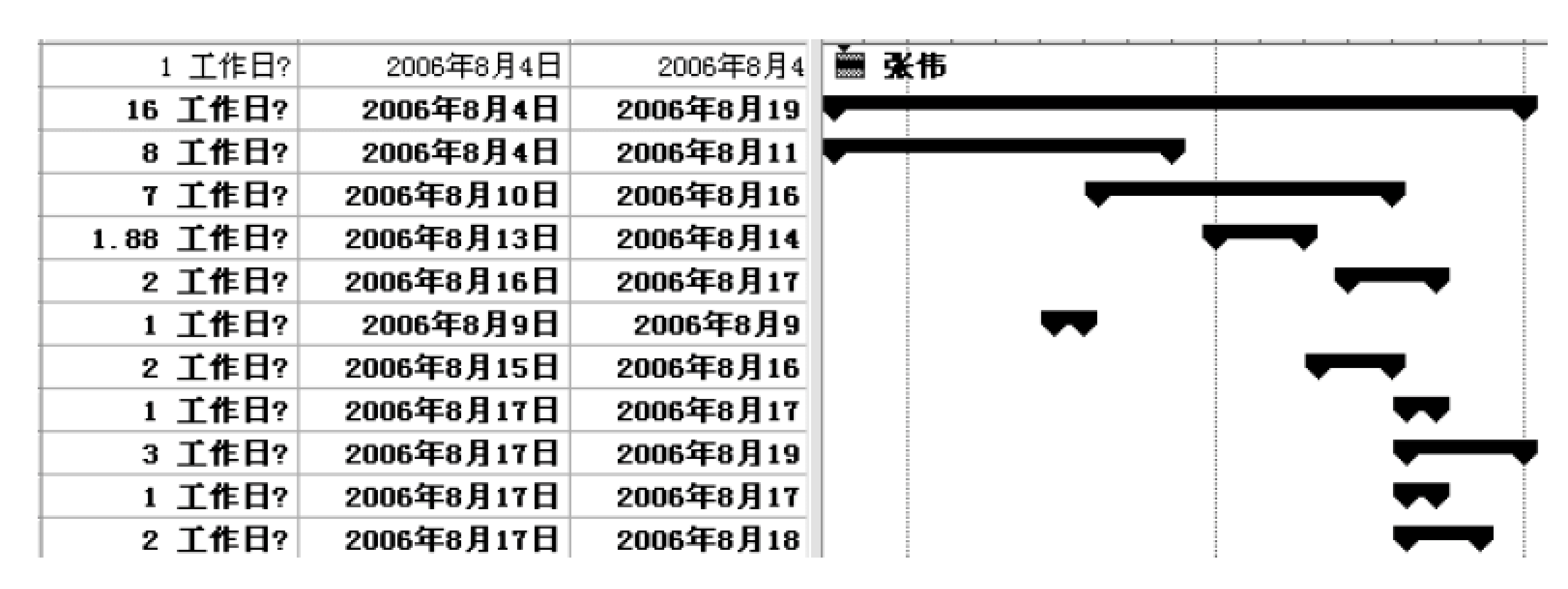
- 关键路径是指一系列决定项目最早完成时间的活动。它是项目网络图中最长的路径,并且有最少的浮动时间。
- 寻找关键路径时,必须首先绘制网络图(WBS——排序),估计每一项任务的历时,然后确定关键路径。

2)计划评审技术(PERT)

- 当具体活动历时估算存在很大的不确定性时,用 PERT 方法估计项目历时。PERT 将关键路径法应用于加权平均历时估算。
- PERT 根据乐观的、最可能的、悲观的活动历时估计进行项目历时估计。公式为

PERT 加权平均=(乐观时间+4×最可能时间+悲观时间)÷ 6

计划任务有很多成熟的管理工具可以应用,其中美国学者甘特发明的一种使用条形图编制项目工期计划的方法,是一种比较简便的工期计划和进度安排工具,称为“甘特图”。甘特图在微软公司出品的 Project(最新版本是 2007)中有完整实现。图 8-17 是甘特图的一个项目应用案例(由 MS Project 根据计划任务自动生成)。



1 工作日?	2006年8月4日	2006年8月4	张伟
16 工作日?	2006年8月4日	2006年8月19	
8 工作日?	2006年8月4日	2006年8月11	
7 工作日?	2006年8月10日	2006年8月16	
1.88 工作日?	2006年8月13日	2006年8月14	
2 工作日?	2006年8月16日	2006年8月17	
1 工作日?	2006年8月9日	2006年8月9	
2 工作日?	2006年8月15日	2006年8月16	
1 工作日?	2006年8月17日	2006年8月17	
3 工作日?	2006年8月17日	2006年8月19	
1 工作日?	2006年8月17日	2006年8月17	
2 工作日?	2006年8月17日	2006年8月18	

图 8-17　甘特图实例

HRMIS 的计划任务安排

有关 HRMIS 的测试 WBS，可使用 MS Project2007 来管理，在此抽样给出 HRMIS 的系统功能部分(如图 8-18 所示)。

⊟ 功能测试	3 工作日?	2008年12月29日	2008年12月31日
⊟ 员工信息管理	2 工作日?	2008年12月29日	2008年12月30日
新增员工信息、修改员工信息和删除员工信息	1 工作日?	2008年12月29日	2008年12月29日
部门调动	1 工作日?	2008年12月29日	2008年12月29日
⊟ 合同管理	1 工作日?	2008年12月29日	2008年12月29日
合同签订	1 工作日?	2008年12月29日	2008年12月29日
合同修改	1 工作日?	2008年12月29日	2008年12月29日
合同删除	1 工作日?	2008年12月29日	2008年12月29日
⊟ 工作履历	1 工作日?	2008年12月30日	2008年12月30日
添加工作履历	1 工作日?	2008年12月30日	2008年12月30日
修改工作履历	1 工作日?	2008年12月30日	2008年12月30日
删除工作履历	1 工作日?	2008年12月30日	2008年12月30日
参加培训记录	1 工作日?	2008年12月30日	2008年12月30日
信息变更历史记录	1 工作日?	2008年12月30日	2008年12月30日
⊟ 培训信息管理	1 工作日?	2008年12月31日	2008年12月31日
培训信息添加、修改和删除	1 工作日?	2008年12月31日	2008年12月31日
培训机构添加、修改和删除	1 工作日?	2008年12月31日	2008年12月31日
培训信息的培训机构变更	1 工作日?	2008年12月31日	2008年12月31日

图 8-18　HRMIS 计划任务抽样

8.3.7　其他特殊要求

在本书第一篇的论述中，已经知道测试工作是介于开发和维护阶段的中间阶段，测试工作需要参阅大量的项目开发资料，并且有时需要提前到项目启动阶段、系统设计阶段，所以列出所需参考或者引用的文档使计划阅读者获得完整全面的信息是十分必要的。

测试计划作为测试的起点，既是对测试工作的规约，也是对开发的一种承诺，所以测试计划中需要标明测试工作的产出物及保存方式，如用例、缺陷等。

测试本身没有明显的停止时间，缺陷总是在不断地被发现被修正。但是项目是有明确的截止时间的，所以需要在测试计划中明确测试的终止条件。指定测试过程正常终止的条件(如测试充分性是否达到要求、缺陷检出率低于多少)，并确定导致测试过程异常终止的可能情况(如接口错误)。

确定测试准入和准出条件，对于关键性测试工作应严格地限制测试的准入准出条件，避免因为测试门槛低，导致缺陷反复，降低测试的效率，导致测试团队成为项目组内被抱怨的对象，损害测试人员的工作热情。测试准出条件是根据企业的质量管理要求和项目的产品质量需求拟定的产品上线前的检定状态，准出条件往往是产品发布的一个必备条件。

8.4 测试计划的编写格式

前面针对测试计划的方方面面做了很多论述，目的是讲明白测试计划是怎么来的，起什么作用，内在机理是怎么样的，要知其然也要知其所以然。那么，这些元素在测试计划中最终是如何体现的呢？测试计划的编写是有章可循的，也就是通俗地说的测试计划有文档模板可以“依着葫芦画瓢”。2008年，国家标准委员会刚刚发布了新版的计算机软件测试文档标准GB/T 9386，其中就有关于测试计划部分的编写要求。GB/T 9386—2008《计算机软件测试文档编制规范》规定了如下的测试计划结构。

1．测试计划标识符

为测试计划规定一个唯一表示符，它用于标识测试计划的版本、等级，以及与该测试计划相关的软件版本

2．引言

概述被测的软件项和软件特征，可以包含系统目标、背景、测试范围、引用文档等。

3．明确测试项

测试项指将要测试的对象，纲领性描述在测试范围内对哪些具体内容进行测试，包括其版本、修订级别等。

4．明确要测试的特征

标识所有要测试的软件特征及其组合，并表示与每个特征或每个特征组合有关的测试设计说明。

5．明确不要测试的特征

对以上确定的要测试的特征中，由于某种原因，无法对其进行测试，应列出不要测试的所有特征及其理由。

6．方法

描述测试的总体方法。对每个主要的特征组或特征组组合，规定要确保这些特征组得到充分测试的方法。规定用于测试指定特征组所需的主要活动、技术和工具。

应详尽的描述方法，以便标识出主要的测试任务，并估计执行各项任务所需要的时间。

规定所希望的最低程度的测试充分性，指明用于判断测试充分性的技术。规定任何

补充的结束准则。

标识对测试的主要约束，例如测试项的可用性、测试资源的可用性和测试截止期限等。

7. 测试项通过准则

确定每个测试项是否通过测试或者测试失败的准则。如每个测试用例都需要一个预期的结果一样，每个测试项同样都需要一个预期的结果。下面是制定测试项通过或失败标准需要考虑的一些例子：

- 测试用例覆盖率；
- 测试用例的通过率；
- 缺陷的数量、严重程度和分布情况；
- 缺陷修复率等。

8. 暂停准则和恢复要求

规定用于暂停与该计划有关的测试项的全部或部分测试活动准则。规定恢复测试时必须重复的测试活动。

常用的测试暂停准则如下：

- 关键路径上的未完成任务；
- 大量的缺陷；
- 严重的缺陷；
- 不完整的测试环境；
- 资源短缺。

9. 测试交付项

规定测试完成后所应递交的文档。

10. 测试任务

标识准备和执行测试所需要的任务集合。标识各项任务间的所有依赖关系和所要求的任何特殊技能。

11. 环境要求

详细说明测试环境必要的和希望的特性。详细内容应包括各种设施的物理特征。这些设施包括硬件、通信和系统软件、使用方式以及支持测试所需的任何其他软件和设备。还应规定这些测试设施、系统软件和专有组成部分（如软件、数据、硬件）所需的安全等级。

标识必要的特殊测试工具及其他任何测试要求，标识测试组目前尚不可用的所有需要的来源。

12. 职责

标识负责管理、设计、准备、执行、监督、检查和解决的各个小组。这些小组可以包括开发人员、测试人员、操作员、用户代表、技术支持人员、数据管理员和质量保证人员。

13. 人员配备和培训要求

按技能等级提出测试人员配备要求。标识为提供必要技能的培训选项。

14. 进度

包括在软件项目进度中标识的测试里程碑以及所有的测试项传递事件。

定义所需要的其他测试里程碑,估计完成每项测试任务所需要的时间,为每项测试任务和测试里程碑规定进度,对每种测试资源(设置、工具、人员)规定使用期限。

15. 风险和应急

标识测试计划的高风险假设,对各种风险提出应急措施。

16. 批准

规定本计划必须由哪些人审批(姓名、职务),为签名和填写日期留出位置。

另外,在编写测试计划时应注意上述各项应按照规定的顺序排列,附加的项可以直接加在批准项之前。如果上述每一项的部分内容或全部内容是在另一个文档里,则可以列出引用材料的出处以代替相应的内容,引用的内容必须附在测试计划里或向该计划的用户提供。

通过对标准中测试计划的编写格式的了解,可以看到在 8.3 节所讲的测试计划的几大要素(测试的质量需求、测试范围的识别、测试策略、计划任务、测试资源、风险管理和一些特殊要求等)构成了测试计划的核心内容,这也正表明 8.3 节和 8.4 节是一脉相承的关系。为了方便对照阅读,将测试计划的编写要素投射到国标中测试计划的编写格式上,可以得到这样一张关系图(如图 8-19 所示),它们具有不谋而合的一一对应关系。

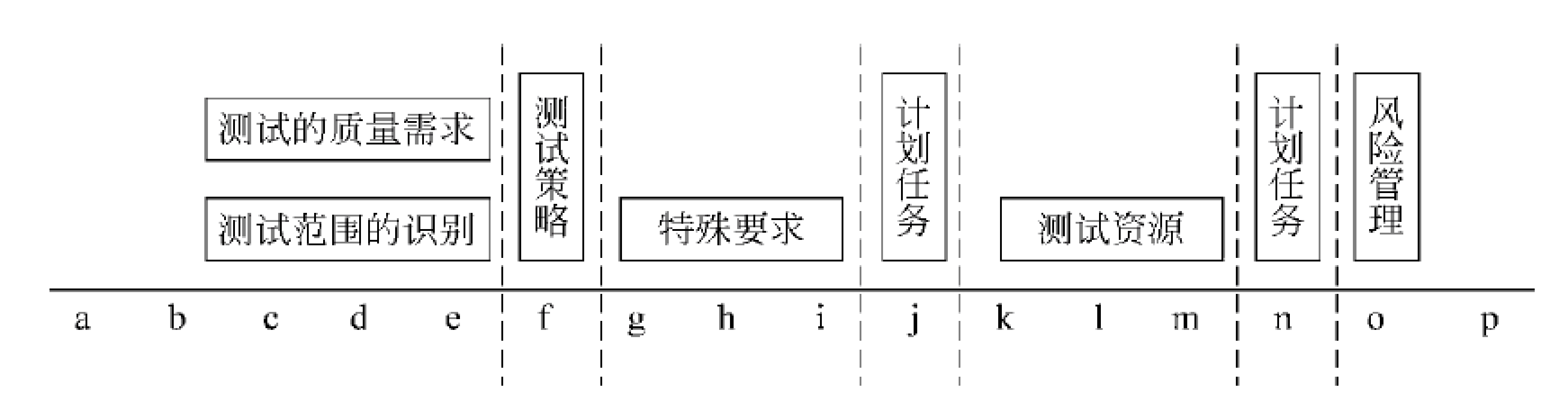


图 8-19 要素对应关系

到本节为止,有关测试计划的内容已经全部讲完,在 8.5 节将给出一些测试计划的实例,通过实例讲解的形式进一步加深理解测试计划的有关内容。

8.5 测试计划实例及点评

HRMIS 是一套面向中小企业的人力资源管理系统,主要解决员工信息管理、合同管理、员工履历管理和培训管理等。针对这个项目的特点和产品需求规格说明,从第三方测试的角度拟制了一份测试计划。

【案例项目测试计划】

人力资源管理系统(HRMIS)测试计划

1. 测试计划标识符

P200901-001-ST-TPL

2. 引言

(1) 目标

制定人力资源管理系统的系统测试计划的目标是:

- 细化准备和进行系统测试所需要的活动。
- 与所有负责方沟通有关他们要执行的任务以及执行任务时所安排的进度。
- 确定用来准备计划的信息源。
- 确定进行系统测试所需要的测试工具和环境。

(2) 背景

要想实现企业快速应变能力,必须能够尽早获得和传输需求信息或变化的信息,必须更快地处理这种变化信息,并更快地做出相应的处理,以满足或处理这种变化。所以人力资源管理系统,也就成为越来越多企业的选择,也是人力资源管理科学化的必然的选择。

筹建的人力资源管理系统立足于当前主流系统基本功能集,并力图在某个领域进行技术创新,提高系统的易用性、稳定性、扩展性。

受××××公司的委托,本系统由××××公司研发,双方于200×年×月×日签订协议,项目正式启动。人力资源管理系统(HRMIS)一期计划实现员工信息管理和培训管理,并且实现多级授权的权限控制。公司内部已于200×年10月进行了项目立项,并在需求调研的基础上制定了开发计划,本测试中心依据开发流程特制定本测试计划,以对HRMIS的相关测试活动做出指导。

(3) 范围

该测试计划覆盖了人力资源管理系统的功能模块,从该系统的功能性、可靠性、易用性、效率、维护性、可移植性及其用户手册等七个方面进行测试和综合的评价。

(4) 引用文档

下列文档用作该测试计划的信息源:

- 人力资源管理系统需求说明书;
- 人力资源管理系统设计说明书;
- 人力资源管理系统开发计划;
- 人力资源管理系统质量保证计划;
- 人力资源管理系统配置管理计划;
- ××公司系统开发标准及规程。

【点评】

本节范围部分描述不够清晰。根据8.3节的分析,可以知道HRMIS的测试范围可以描述为:

分析 HRMIS 的系统需求和设计，HRMIS 的质量需求概览如下：

分类	内容描述			遗留问题
结构	易用性、稳定性、扩展性			
	界面风格保持统一			
功能	管理功能	系统访问权限管理		
		用户管理		
		部门管理		
	业务功能	员工基本信息管理	基本信息管理	
			工作履历管理	
			培训实施记录	
			合同管理	
			部门调动	
			信息变更历史记录	
		考勤/考评管理		
		员工工资管理		
		培训管理	培训信息管理	
			培训机构管理	
	接口	考勤系统		接口需求及设计
		财务管理系统		接口需求及设计
		物流管理系统		接口需求及设计
		生产管理系统		接口需求及设计
		CRM		接口需求及设计
性能	系统登录	15User并发，平均响应时间<10s		
	考勤高峰	15User并发，平均响应时间<10s		
	信息修改	平均时间小于15s		确定典型业务
	数据转换和传送	平均时间小于15s		确定典型业务
其他要求	不要求跨平台			
	故障恢复			
	涉及时间的要求精确到日期，不要求精确到时间			
	培训费用精确到元角分，计费币种为人民币			

说明：粗体是跨出原系统需求的新增测试需求；阴影标注区是需要进一步确认的测试需求。

（1）功能测试

一级模块	二级模块	三级模块
员工信息管理	新增员工信息	
	修改员工信息	
	删除员工信息	
	部门调动	
	合同管理	合同签订
		合同修改
		合同删除
	工作履历	添加工作履历
		修改工作履历
		删除工作履历
	参加培训记录	
	培训总结	
	培训考核	
	信息变更历史记录	
培训信息管理	培训信息添加、修改和删除	
	培训机构添加、修改和删除	
	培训信息的培训机构变更	

续表

一级模块	二级模块	三级模块
系统管理	系统用户添加、修改和删除	
	系统用户权限设置	

（2）性能测试：

关键业务清单：

- 系统登录；
- 更新处理；
- 数据转换和传送。

据此，从质量特性角度可以确定 HRMIS 的测试范围如下：

软件质量特性	子特性	是否测试	软件质量特性	子特性	是否测试
功能性	适合性	Y	效率	时间特性	Y
	准确性	Y		资源特性	Y
	互操作性	Y		依从性	Y
	安全保密性	Y	维护性	易分析性	Y
	依从性	Y		易改变性	Y
可靠性	成熟性	Y		稳定性	Y
	容错性	Y		易测试性	Y
	易恢复性	Y		依从性	Y
	依从性	Y	可移植性	适应性	Y
易用性	易理解性	N		易安装性	N
	易学性	N		共存性	Y
	易操作性	N		易替换性	Y
	吸引性	N		依从性	Y
	依从性	N			

（3）非测试项说明

根据产品当前所处的阶段，在首轮测试期间对产品的易用性不予测试，因 HRMIS 的使用环境无跨平台方面的要求，因此对于产品可移植性中的适应性不做测试。

另外，在 HRMIS 的一期建设中，系统接口尚处于调研或者研发初期阶段，故本测试计划对于质量需求中所定义的接口暂不测试。

3. 测试依据

- GB/T 16260.1—2006《软件工程 产品质量 第 1 部分：质量模型》；

- GB/T 16260.2—2006《软件工程 产品质量 第2部分：外部度量》;
- ××××公司测试规程;
- 人力资源管理系统需求说明书;
- 人力资源管理系统设计说明书。

4. 方法

测试人员应根据系统文档准备所有的测试设计、用例以及规程说明。公司人力资源部门应协助开发测试设计和测试用例，这样做有助于确保测试能体现系统的实际使用。

(1) 文档测试

应对软件文档的正确性、完整性、一致性、易浏览性进行逐一检查和验证。

(2) 功能性测试

通过采用用户文档中规定的方式和流程执行软件，检查软件输出结果和相应的执行过程以及其功能特性是否满足相应的要求，是否正确，是否具有相应的安全措施。

(3) 可靠性测试

分析出不符合业务逻辑的操作过程和数据，采用软件文档中未规定和不允许的方式和数据执行软件，必要时会采用破坏性测试，如断电、断网、非法关机等手段，以检查软件的执行过程、方式和结果，验证其容错、健壮、错误恢复能力。

(4) 易用性测试

通过检查软件以及软件执行过程中的界面、图形、文字、信息和标识是否容易理解，易于浏览，检查软件的输入、操作方式是否便捷、易用、易学。

(5) 效率测试

利用性能测试工具软件，分别模拟系统软件所设计的用户数量和系统软件所能承受的用户数量，对软件处理能力和数据传输能力进行测试。并验证软件在不同状态下的系统响应时间、吞吐量、资源利用率等指标和规定要求的符合性。

(6) 回归测试

为了测试在系统期间做过的程序修改，应对系统进行若干次重复测试。对系统的每一新版本应做一次回归测试，从而检测由于程序修改所导致的意想不到的影响。

【点评】

测试方法栏描述过于简略，难以起到对实际测试活动的指导作用。从8.3节的描述中，我们知道更为完整的方法描述还应该加上对应测试类型所采取的测试策略，比如：

(1) 单元测试策略

测试目标	确保所测试的单元功能正常，没有冗余语句
测试类型	单元测试
测试范围	模块：所有中间处理逻辑单元 质量特性：功能性(适合性、准确性)、维护性

续表

技术描述	使用 CppUnit 开发、执行测试用例,检验单元提供的方法 使用 Logiscope 对单元的过程调用关系、函数控制流进行分析 使用 Rational TestRealTime 对单元进行语句覆盖分析
开始标准	模块代码编写完成,调试无误
完成标准	所预定的测试覆盖(语句覆盖、路径覆盖等)达到目标值; 所预定的测试用例执行通过
测试重点和优先级	关键模块的处理逻辑单元优先
需考虑的特殊事项	测试工具对编程语言的支持度 驱动模块和桩模块的设计和编制

(2) 集成测试策略(略,参见 8.3.4 节)

(3) 系统测试策略(略,参见 8.3.4 节)

另外,在有些测试组织,方法栏有时还会依据被测系统的业务流程进行测试操作步骤的描述。

5. 测试项通过准则

略。

6. 暂停准则和恢复要求

(1) 暂停准则

如果测试中发生严重缺陷,导致 50%的测试用例无法执行,则将测试暂停。

(2) 恢复要求

出现测试暂停后,当系统的新版本向测试组传递时,应执行回归测试。

7. 测试交付项

系统测试期间可能形成的文档列示如下,这些文档在测试结束后进行归档。

- 系统测试计划;
- 系统测试说明;
- 系统测试报告;
- 系统测试记录;
- 系统测试缺陷报告。

8. 测试任务

任务列表

任　　务	前期任务	特殊技能	责　　任	投　　入	完成日期
(1) 准备测试计划	结束人力资源管理系统设计描述和初步的开发计划	无	测试项目负责人 测试分析员	4(人日)	—
(2) 准备测试设计说明	任务(1)	通晓公司的人力资源管理规程	测试分析员	8	—

续表

任务列表					
任　　务	前期任务	特殊技能	责　　任	投　　入	完成日期
(3) 准备测试用例说明	任务(2)		测试设计员	6	—
(4) 准备测试规程说明	任务(3)	—	测试设计员	4	—
(5) 建立最初的员工信息数据库	任务(4)	—	测试设计员	2	—
(6) 结束测试项传递并向测试组传递该公司的人力资源管理系统	结束集成测试	—	开发项目组经理	—	—
(7) 检查执行该系统需要的所有的工作控制规程	任务(6)	工作控制经验	测试分析员	1	—
(8) 组装并连接该公司人力资源管理系统	任务(6)	—	测试员	1	—
(9) 执行测试	任务(5)、(8)	—	测试员	6	—
(10) 检查测试结果	任务(9)	通晓公司的人力资源管理的需求	测试设分析员	2	—
(11) 解决测试事件报告	任务(9)、(10)	—	测试项目负责人 开发组经理 人力资源部经理	3	—
(12) 重复(6)～(11)直到达到测试通过标准。	任务(11)	—	—	4	—
(13) 编写系统测试报告	任务(12)	—	测试项目负责人 人力资源部经理	2	—
(14) 将所有测试文档集和测试数据传输给配置管理组	任务(13)	—	系统测试组 配置管理员	1	—

9. 环境要求

(1) 硬件

- 测试应在××公司的硬件配置下进行，应满足如下最低配置要求；
- 服务器：处理器 Xeon 2.0GHz×2\内存 2GB 以上\硬盘 160GB 以上；
- 客户端：处理器 Pentium 4 1.1GHz\内存 512MB 以上\硬盘 80GB。

(2) 软件

- 服务器端、客户端操作系统：Microsoft Windows 2000/XP/2003；

- 服务器端数据库：MySQL 6.0。

(3) 工具

性能测试工具采用 LoadRunner 9.1

【点评】

环境部分最好可以引用有关系统设计说明的拓扑结构图，有助于测试计划的接收人更好地理解项目对测试环境的要求。本案 HRMIS 的测试环境请参考第 9 章测试执行部分的有关内容。

10. 职责

系统测试组：对测试及技术测试业务进行全面管理。

公司人力资源部：该组是公司人力资源管理系统的终端用户，在审查测试设计说明、执行测试等活动中应协助系统测试组工作。

项目开发组：传递要测试的系统，并响应系统测试事件报告。该组对需要排错的任何程序进行调试。

11. 人员配备、培训要求和资源配置

(1) 人员配备

测试组需要下列人员开展测试项目：

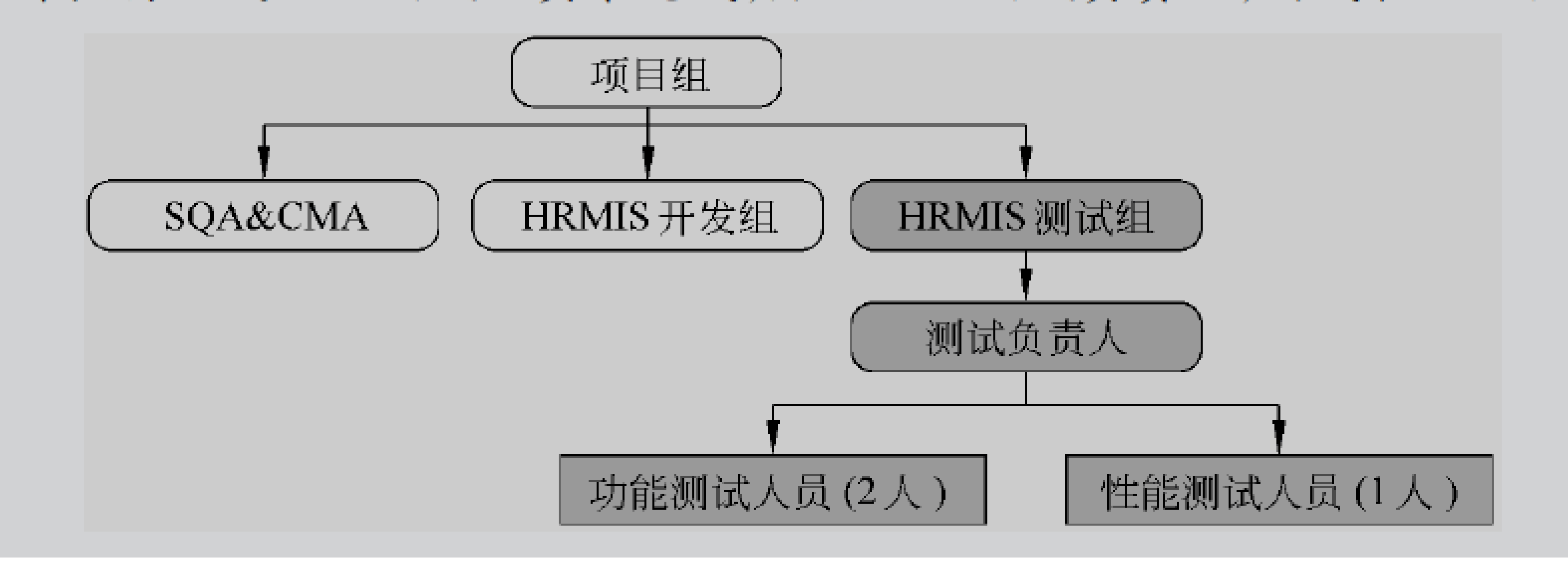
- 测试项目负责人一名；
- 测试设计员一名；
- 测试执行人员一名；
- 性能测试员一名；
- 测试系统管理员一名(可共享)。

(2) 培训

公司人力资源管理部门的人员必须经过培训，以便对数据录入事务进行处理。用户文档作为培训的基础。

【点评】

在此直接给出 8.3 节的有关结果：HRMIS 的整体测试工作按阶段可以分为单元测试、集成测试和系统测试三个主要的阶段，其中单元和集成测试计划由开发人员自行完成，故这部分工作基本上不存在人力资源支出。考虑到系统测试工作量和项目周期，系统测试工作共需二或三名测试人员来完成，另设一名测试负责人，构成和组织框架如下：



（1）测试时间

测试类型	起始时间	结束时间
单元测试	20××年12月10日	20××年12月16日
集成测试	20××年12月18日	20××年12月29日
系统测试	20××年12月29日	20××年1月19日

（2）测试设备和工具

根据HRMIS的结构，暂时为其配置3台测试机，其中一台作为数据库服务器使用。测试工具使用SVN配置管理工具、TestDirector 8.0和LoadRunner 9.1，其中SVN可以与项目组共享使用，TestDirector属于公司的公共平台，不需要另行配置。

12. 进度

见测试任务表。

【点评】

通常在做测试计划时以所分析出的测试需求和测试范围为依据，进行科学的WBS，以方便管理和进度控制。通过8.3节，知道本案的测试计划可以借助MS Project 2007来生成，有关内容请参考*HRMIS Test&Dev Plan*（见附属光盘，.mpp文件格式）。

13. 风险和应急

- 如果系统故障严重影响测试进度，开发经理应分派一名全职人员到测试组做调试工作。
- 如果一位监管人员对于测试工作不够用，人力资源部经理应确定第二位监管人员。
- 如果硬件出现的问题影响系统在白天的使用，则测试组应安排其夜晚的活动。

【点评】

在此直接给出8.3节的有关结果，风险评估表如下：

<table>
<tr><td>编　　号</td><td colspan="2">—</td><td>填表人</td><td>—</td><td>填表日期</td><td>—</td></tr>
<tr><td>项目名称</td><td colspan="4">人力资源管理系统(HRMIS)系统测试</td><td>项目经理</td><td>—</td></tr>
<tr><td>风险编号</td><td>风险名称</td><td>风险类别</td><td>发生概率</td><td>产生后果</td><td colspan="2">风险等级</td></tr>
<tr><td>1</td><td>性能测试工具掌握程度</td><td>技术风险</td><td>很可能</td><td>实质性危害</td><td colspan="2">A</td></tr>
<tr><td>2</td><td>系统架构评估标准认可</td><td>技术风险</td><td>可能</td><td>显著危害</td><td colspan="2">B</td></tr>
<tr><td>3</td><td>第三方系统承建方支持度</td><td>外部可预测风险</td><td>可能</td><td>显著危害</td><td colspan="2">B</td></tr>
</table>

针对不同风险的特点，需要制定一些有效措施加以规避（或者转嫁），在本例中可以制定以下措施：

风险 1：性能测试工具掌握程度

可以采取的措施：

- 安排外部培训
- 租借有经验的人员加入项目组

风险 2：系统架构评估标准

可以采取的措施：

- 客户、系统承建方和测试实施单位共同拟定评估标准
- 听取有经验的系统架构师的建议

第三方系统承建方支持度

可以采取的措施：

- 列出支持项，递交给客户，争取主动权
  争取客户的系统管理人员作为备份

14. 批准

测试项目负责人：　　　　日期：

开发项目经理：　　　　日期：

质量保证经理：　　　　日期：

8.6 测试计划的最佳实践

前面从理论高度和标准角度对测试计划做了全面论述，这对于一个初入软件测试行业的人员来说是必要的。就好比一个要学功夫的人，一开始就投身于一个名门正派，打下一个良好根基对于他的未来发展是至关重要的。

当然，只有理论的高度而没有实践的深度也是不足取的。我国的信息产业发展远远落后于国外发达国家，甚至与印度也有不小的差距。确切的说，我们仍是处于一个学习阶段，引入阶段，面对这么多泊来品，如何消化吸收，如何根据整体的 IT 产业环境构建适合的测试之道，是时代提出的对软件从业人员的素质要求。

测试计划不一定要尽善尽美，但一定要切合实际，要根据项目特点、企业实际情况来编制，不能脱离现实，测试计划的制定应与当前企业或者部门、甚至项目组的研发流程相适应，比如迭代式开发，测试计划也应该是一个“迭代”的过程，而且要不断的跟踪监控计划的执行情况。测试计划是用来执行的不是拿来供的，不要吝于对测试计划的修改，软件需求、软件开发、人员组成等都是经常发生变化的因素，测试计划也要根据实际情况的变化而不断进行调整，以满足实际测试要求。

测试计划的计划任务建议使用 Project 或者其他一些计划任务管理工具，无须拘泥于形式。另外在做计划时，估算资源、时间、预算等要留有一定余地，以保留应对各类风险的

主动性。

计划完成之后要进行评审，以听取不同层次的人员的意见。关于测试计划的评审可以从以下几个方面去考虑：

① 项目需要遵循什么样的国际标准或者客户的有关特定要求，哪里能找到这些文件？

② 哪种测试计划已被制定，计划中定义的测试活动怎么融合在整个项目计划中？

③ 测试需要什么特殊软硬件设备支持？

④ 对于由客户或者第三方提供的软件、硬件应用什么评估流程，以确保设备的适用性和测试结果的公正性。

⑤ 文档是否符合相关标准？（如格式、编写规范等）

⑥ 一些必要的输出文件是否在计划中有充分的说明？

⑦ 测试策略是否满足要求，说明是否充分？

⑧ 每种测试是否充分考虑了可能的变更？

⑨ 缺陷的类型和严重程度是否有说明？

本章小结

万事开头难，软件测试伊始也非一路坦途。从开发过渡到测试不仅仅是工作内容的转换，还有思维角度的转换。为了软件测试后续工作的开展，需要一开始就要防微杜渐，把好关，铺好路，为自己赢得足够的工作空间。

本章详细描述了项目启动期间软件测试的主要工作，如资料审查、软件移交，同时对于测试计划的重要性和制定方法、步骤等也做了较为详尽的说明。测试计划需要进行测试质量需求分析、风险分析、测试范围的识别、测试策略的制定、测试资源评估和计划任务的安排，这些构成了测试计划的主体。

测试计划的制定本质上是一个逐渐理清测试任务的过程，其间也有一些测试分析的成分，但是都属于较浅层次的。第 9 章将进入软件测试生命周期的第二个阶段“测试分析”。第 9 章将重点讲解一下软件测试需要做哪些分析工作，回答诸如如何从产品需求规格说明书导出测试需求、测试需求如何管理等问题。

CHAPTER

第 9 章

测 试 分 析

软件开发过程中的需求工程已经深入人心，获得了广泛的认可，但是对于软件测试中的需求分析却很少提及，在很多测试团队中，测试需求分析也往往成了被忽略的对象。但是殊不知，软件测试需求对于一次成功的软件测试是至关重要的。具有良好的软件测试需求分析习惯的测试团队才具备可持续的测试能力改进的基础。

9.1 什么是软件测试需求

软件测试需求无非就是解决“测什么”的问题，但是仅仅这样认为的简单化处理对于测试需求来说是不全面的，这种思维习惯也会极大地影响后续有关于软件测试需求的进一步挖掘。那么，如何来准确的描述软件测试需求呢?

从软件测试需求的作用和后续测试工作的需要来看，软件测试需求大体可以从以下几个方面来阐释：

- 测试需求的核心就是来指明被测对象中什么需要测试，也就是“测什么”。这个“什么”内容很广泛，可以是功能，也可以是业务流程或者是性能、易用性等；
- 测试需求要全部覆盖已定义的业务流程以及功能、非功能需求；
- 制定的测试需求项必须是可核实的。即，它们必须有一个可观察、可评测的结果。无法核实的需求不是测试需求；
- 测试需求应指明满足需求的正常的前置条件，同时也要指明不满足需求时的出错条件，这样有利于在后续的测试执行过程中对被测对象从正、反两个方面进行全面的测试；
- 制定的测试需求中不涉及测试数据，这属于“怎么测”的问题范畴，这个留到测试设计这个环节去解决。

测试需求是制定测试策略、测试用例设计和开发的基础，测试需求分析分解的越详细精准，则表明对所测软件的了解越深，所要进行的任务内容就越清晰，对测试用例的设计质量的帮助越大，就更有把握保证测试的质量与控制测试计划任务的进度。详细的测试需求还是衡量测试覆盖率的重要指标，如果没有详细的测试需求就很难对测试覆盖情况进行有效的计算和评估。

9.2 测试需求分析过程

测试需求是整个测试过程的基础，在测试活动中，首先需要明确测试需求，才能决定怎么测。如果把测试活动比作软件生存周期，测试需求分析就相当于软件的需求分析，测试用例设计相当于软件的概要设计和详细设计，测试执行相当于软件的编码过程。只是在测试过程中，把“软件”两个字全部替换成了“测试”。这样，就明白了整个测试活动的依据来源于测试需求。

测试需求分析活动可以分为测试需求采集、测试需求分析、测试需求评审三个子活动，分析过程可以用图 9-1 表示。

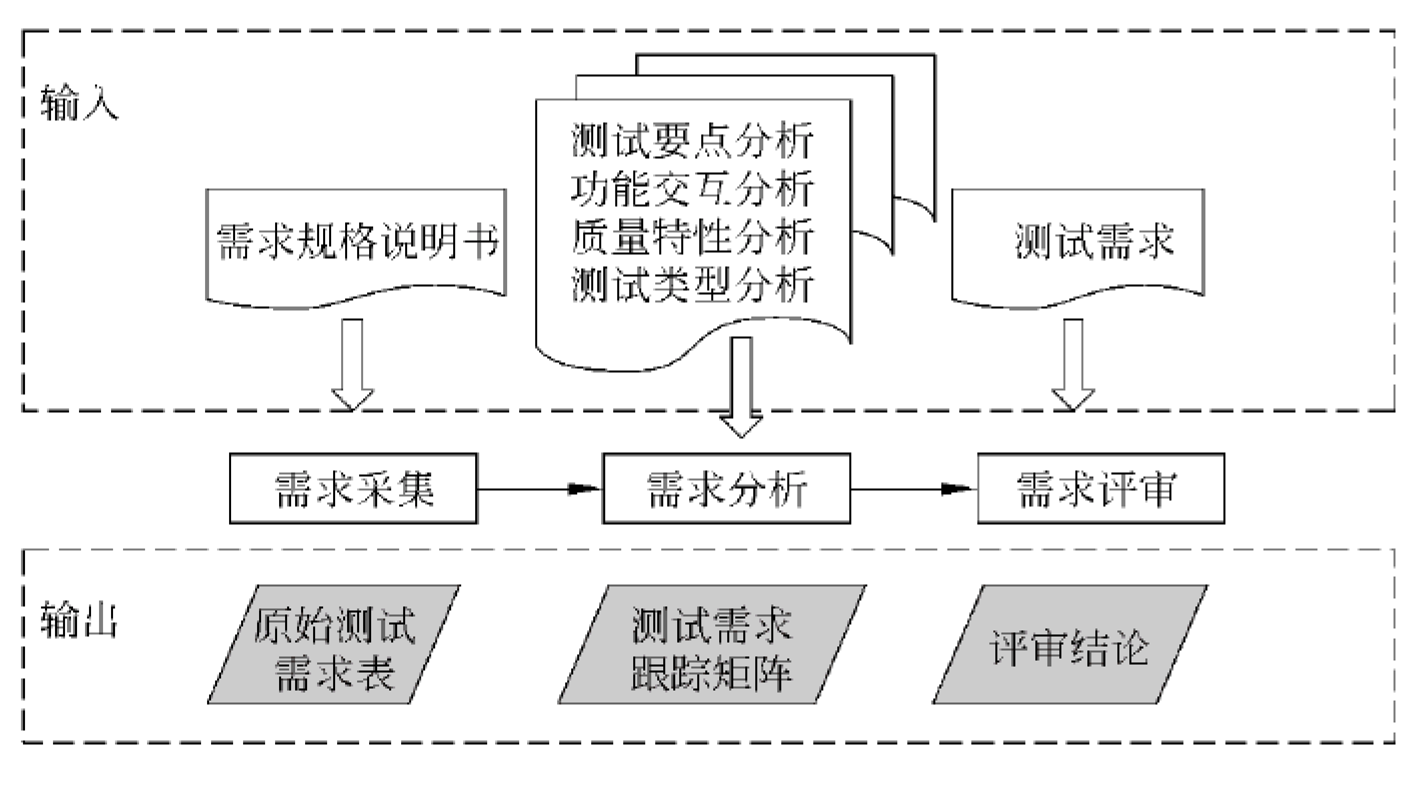


图 9-1　测试需求分析活动

9.2.1 需求采集

理论上，测试需求通常是以待测软件的软件需求和设计文档为基础进行分析而转变过来的，然而有些时候，由于需求分析工作不够完善和对系统缺乏全面认识等原因，软件需求规格说明往往无法完整、准确、具体地反映用户的需求，尤其是对性能、可靠性、安全性和系统约束等内容缺乏描述，因此，基于这种软件需求规格说明来对软件进行测试验证是不客观和不可信的。在这种情况下，测试人员必须在现有的软件需求规格说明的基础上做进一步的测试需求开发，形成用户、开发人员和测试人员一致认可的测试需求，并在此基础上开展测试工作。

软件测试需求与测试所处的测试阶段有关。在单元测试和集成测试阶段，软件测试需求主要来源于详细设计说明书、概要设计说明书，有些还需要从软件代码中搜集和提取，通过对其技术结构和功能结构的分析获得原始测试需求。在系统测试和验收测试阶段，软件测试需求主要来源于需求规格说明书，必要时需要参考设计文档。

原始测试需求指的是需求来源中的那些具有可测试性的需求或特性。即这些需求或特性必须存在一个可以明确预知的结果，可以用某种方法对这个明确的结果进行判断、验证，验证是否符合文档中的要求。如果对于某条需求或某个特性，无法通过一个明确的方

法来进行验证,或者无法预知它的结果,那么就意味着这条需求不具备可测试性,该描述存在缺陷,应该请需求分析人员对需求文档进行修改或补充。

原始测试需求的采集过程是测试需求分析阶段首要进行的活动,这个活动的主要目的是提取基本的测试需求。可用表格的形式对软件需求进行梳理,将每一条软件需求对应的开发文档及章节号作为软件需求标识。使用软件需求的简述作为原始测试需求描述,没有文档来源的软件需求可用隐含需求或遗漏需求进行标识,同时还要标明软件需求获取的来源信息,如开发文档、相关标准、与用户或开发人员的交流等。

提取的原始测试需求中,可能存在重复和冗余,所以在提取原始测试需求过程中,可以通过以下方法整理原始测试需求:

① 删除。删除原始测试需求表中重复的、冗余的含有包含关系的原始测试需求描述。

② 细化。对过于简略的原始测试需求描述进行细化。

③ 合并。如果有类似的原始测试需求,在整理时需要对其进行合并。

例如,在本书案例 HRMIS 的测试中,通过阅读分析其需求说明书和有关标准可以得到以下"'人力资源管理系统'原始测试需求"(见表 9-1)。

表 9-1 HRMIS 原始测试需求

“人力资源管理系统”原始测试需求表				
序号	软件需求标识		原始测试需求描述	信息来源
1	3.6.1 培训信息维护	增加培训信息	一条完整的培训信息包括培训的主题、证书、内容、起止时间、费用、地点、机构,其中培训的主题、内容、起止时间、费用、机构为必填项。培训的起始时间不能晚于截止时间,培训费用精确到元角分。每一个输入项的数据规格应遵循数据字典的要求	《人力资源管理系统业务需求说明书》
2	3.6.2 培训机构维护	增加培训机构信息	一条完整的培训机构包括培训机构名称、办公地址、联系人……。机构名称不允许重复	《人力资源管理系统业务需求说明书》
3	3.2.2 时间特性要求		15 个并发用户下,系统登陆的平均响应时间小于 10s	《人力资源管理系统业务需求说明书》
4	隐含需求:在使用中操作错误的易恢复性		程序应对关键数据的操作给出警告或在执行前要求确认	GB/T 17544-1998《信息技术 软件包 质量要求和测试》
⋮	⋮		⋮	⋮

9.2.2 测试需求分析

测试需求分析是测试需求分析过程中最为重要的一个环节,是对原始测试需求进行测试分析,并对分析结果进行合理的测试分解的过程,最终形成产物是测试需求跟踪矩阵。测试需求分析的步骤可用图 9-2 表示,首先针对采集的原始测试需求,分析出测试要

点，然后针对这些测试要点，参考质量特性定义，得出软件的质量特性需求，再从测试类型出发，确定出软件需要进行的测试类型，最后得到测试需求跟踪矩阵。

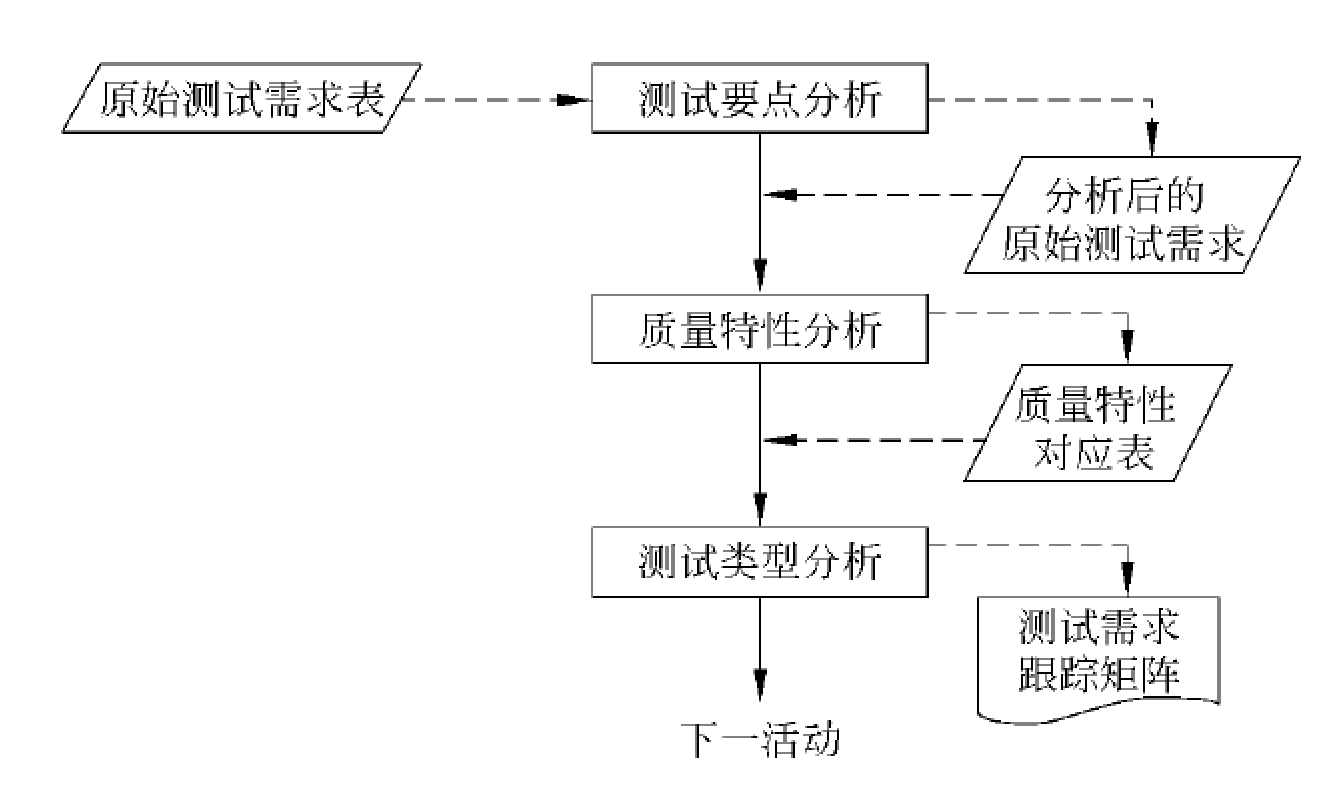


图 9-2　测试需求分析过程

1. 分析质量特性

建立了原始测试需求表后，需要对表中的软件需求进行分解，形成可测试的分层描述的软件需求，称为测试要点。测试要点是通过分析每条测试需求描述中的输入、输出、处理、限制、约束等，给出对应的验证内容。测试要点是测试用例设计的依据，针对测试要点进行用例设计，每个测试要点基本对应一个或多个测试用例。

需求分解需要从两个方面考虑：

① 需求的完整性，经过分解获得的需求必须能够充分覆盖软件需求的各种特征（包括隐含的特征），每个需求必须可以独立完成有意义的功能或功能组合，可以进行单独测试。

② 需求的规模，每个最低层次的需求能够使用数量相当的测试用例来实现，也即测试的粒度是均匀的。

还是以 HRMIS 的测试为例，对于 HRMIS 的"增加培训信息"这个功能点而言，针对该功能输入、输出、处理、限制和约束等方面的描述，可以从正确性验证、数据规范检查、数据关系检查、操作顺序检查等方面直接分析出测试要点，如表 9-2 中标识为 1～6 的测试要点；同时也可以深入分析功能项隐含的测试需求，可以从界面检查、提示信息检查等方面进行分析，如下表标识为 8～11 的测试要点；此外，也有必要根据经验来补充一些容易发生错误的特殊情况，例如本例中培训记录重复保存没有必要，且容易导致系统在使用培训记录时产生信息不一致的错误发生，因此本例中需要验证系统对培训记录重复保存问题的处理，标识为 7 的测试要点见表 9-2。

表 9-2 所举案例的分析主要是针对单条功能项进行的分析，但绝大多数软件产品的功能都不是独立的，功能之间存在交互，因此还需要进行功能交互分析。进行功能交互分析可以防止交互功能的遗漏，提高测试的完备性。功能交互分析是对功能测试方面的分析，通过分析各个功能模块之间的业务顺序，和各个功能模块之间传递的信息和数据，从而得到相应的测试要点。其具体步骤如下（如图 9-3 所示）：

① 在采集的原始测试需求中,确定出原始功能需求。
② 利用步骤①确定的原始功能需求进行交互分析。
③ 将步骤②中得到的交互分析结果列入“原始测试需求表”。
④ 对表中的软件需求进行分解,分析出测试要点。

表 9-2　测试要点分析实例

原始需求描述	标识	测 试 要 点
一条完整的培训信息包括培训的主题、证书、内容、起止时间、费用、地点、机构,其中培训的主题、内容、起止时间、费用、机构为必填项。培训的起始时间不能晚于截止时间,培训费用精确到元角分。每一个输入项的数据规格在数据字典中可以得到	1	输入符合字典要求的各信息后执行保存,检查保存是否成功
	2	检查每个输入项的数据长度是否遵循数据字典的要求
	3	检查每个输入项的数据类型是否遵循数据字典的要求
	4	检查“培训费用”是否满足规定的精度要求
	5	检查在培训的起止时间早晚于截止时间时,所增加的记录是否保存成功
	6	检查“培训主题”、“培训内容”、“起止时间”、“培训费用”、“培训机构”是否为必填项
	7	验证系统对数据重复的检查
	8	针对页面中文字、表单、图片、表格等元素,检查每个页面各元素的位置是否协调,各元素的颜色是否协调,各元素的大小比例是否协调
	9	页面信息内容显示是否完整
	10	检查是否有功能标识,功能标识是否准确、清晰
	11	最大化、最小化、还原、切换、移动窗口时是否能正常的显示页面

第一篇已经对 GB/T 16260—2006《软件工程 产品质量》的有关内容做过介绍,并且也知道软件测试是为软件质量服务的,测试是手段,质量是目的。用户对软件的各种需求本质上是对软件质量的一种诉求,理论上均可以对应到功能性、可靠性、易用性、效率、维护性、可移植性等软件产品的六个质量特性上。因此,为了对软件产品准确地进行质量评价,分析出测试要点之后,需要进一步确定出其所对应的软件质量特性。

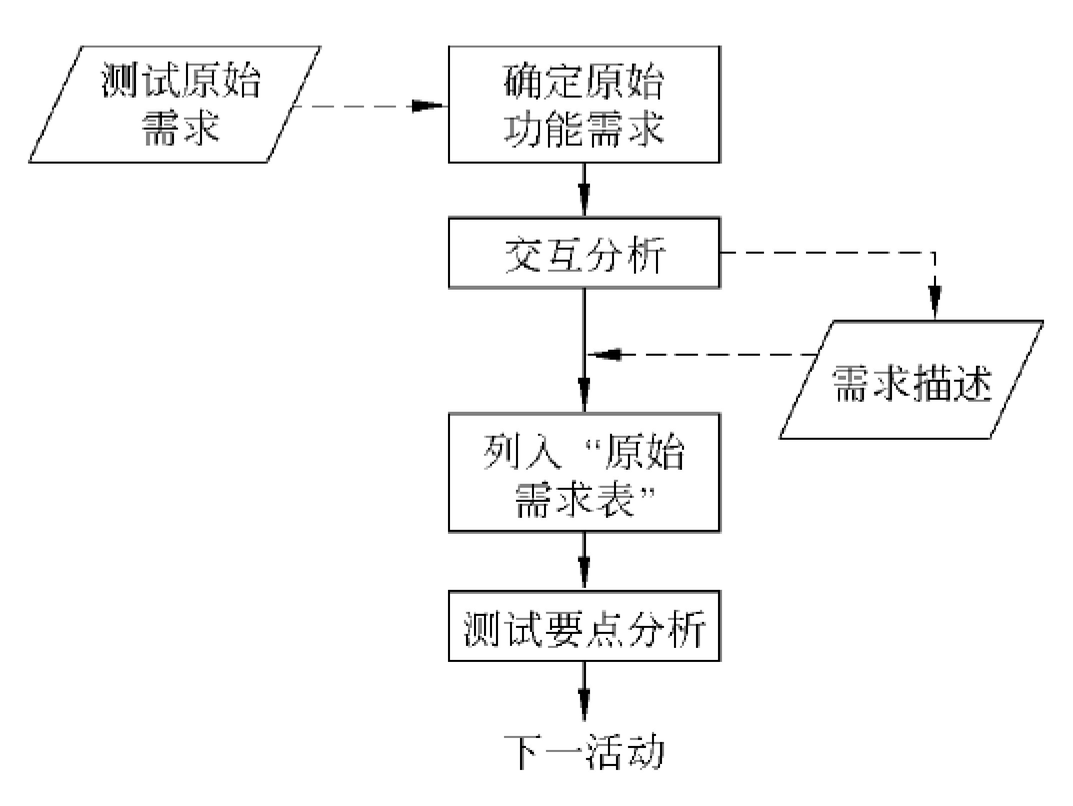


图 9-3　要点分析步骤

先来看一下标准中对有关部分质量特性的定义:

1) 功能性

- 适合性:软件产品为指定的任务和用户目标提供一组合适的功能的能力。
- 准确性:软件产品提供具有所需精度的正确或相符的结果或效果的能力。

2）可靠性

- 容错性：在软件出现故障或者违反其指定接口的情况下，软件产品维持规定的性能级别的能力。

3）易用性

- 易理解性：软件产品使用户能理解软件是否合适以及如何能将软件用于特定的任务和使用条件的能力。
- 易操作性：软件产品使用户能操作和控制它的能力。

根据标准中对特性、子特性的定义，可以比较容易地分析出各测试要点对应的质量特性、子特性，从而确定出软件所应具备的软件质量属性。

通过前面的分析，已经获得了 HRMIS 的部分测试要点，结合 GB/T 16260—2006《软件工程 产品质量》的有关质量特性的定义，可以得到这些测试要点的质量特性的分析结果（见表 9-3）。

表 9-3　HRMIS 的质量特性分析实例

质量特性对应表			
原始需求描述	标识	测试要点	质量特性
一条完整的培训信息包括培训的主题、证书、内容、起止时间、费用、地点、机构，其中培训的主题、内容、起止时间、费用、机构为必填项。培训的起始时间不能晚于截止时间，培训费用精确到元角分。每一个输入项的数据规格在数据字典中可以得到	1	输入符合字典要求的各信息后执行保存，检查保存是否成功	功能性/适合性
	2	检查每个输入项的数据长度是否遵循数据字典的要求	功能性/适合性、可靠性/容错性
	3	检查每个输入项的数据类型是否遵循数据字典的要求	功能性/适合性、可靠性/容错性
	4	检查“培训费用”是否满足规定的精度要求	功能性/准确性
	5	检查在培训的起止时间早晚于截止时间时，所增加的记录是否保存成功	功能性/适合性
	6	检查“培训主题”、“培训内容”、“起止时间”、“培训费用”、“培训机构”是否为必填项	功能性/适合性
	7	验证系统对数据重复的检查	功能性/适合性
	8	针对页面中文字、表单、图片、表格等元素，检查每个页面各元素的位置是否协调，各元素的颜色是否协调，各元素的大小比例是协调	易用性/易操作性
	9	页面信息内容显示是否完整	易用性/易操作性、易理解性
	10	检查是否有功能标识，功能标识是否准确、清晰	易用性/易理解性
	11	最大化、最小化、还原、切换、移动窗口时是否能正常的显示页面	易用性/易操作性

2. **分析测试类型**

不同类型的测试从不同的角度分析和测试产品，会发现不同类型的缺陷。不同产品对应的测试类型集合不一样，每种测试的测试方法和侧重点也有所不同，因此需要对所确定的质量特性进一步分析以确定适合的测试类型，确定需要对被测软件产品采取的测试策略。

从第二篇的学习中我们知道根据测试内容的不同，软件测试可以划分为以下测试类型：功能测试、安全性测试、接口测试、容量测试、完整性测试、结构测试、用户界面测试、负载测试、压力测试、疲劳强度测试、恢复性测试、配置测试、兼容性测试、安装测试等。

根据质量特性的定义，以及各测试类型的测试内容，可以分析出质量特性与测试类型的对应关系。例如，配置测试是确保在不同的硬件和软件配置上实现预期的测试目标功能，对应了可移植性中的适应性。综合 GB/T 15532：2008《计算机软件测试规范》的有关内容以及相关研究成果，给出了软件质量子特性和测试类型的对应关系基准表(见表 9-4)。

表 9-4 质量特性与测试类型对应关系

质量特性分类	质量子特性分类测试内容
功能性	适合性方面
	准确性方面
	互操作性方面
	安全保密性方面
	功能性依从方面
可靠性	成熟性方面
	容错性方面
	易恢复性方面
	可靠性依从方面
易用性	易理解性方面
	易学性方面
	易操作性方面
	吸引性方面
	易用性依从方面
效率	时间特性方面
	资源利用方面
	效率依从性方面
维护性	易分析性方面
	易改变性方面
	稳定性方面
	易测试性方面
	维护性依从方面
可移植性	适应性方面
	易安装性方面
	共存性方面
	易替换性方面
	可移植性依从方面

对应关系

测试类型
功能测试
安全性测试
接口测试
容量测试
完整性测试
结构测试
用户界面测试
负载测试
压力测试
疲劳强度测试
恢复性测试
配置测试
安装性测试
兼容性测试

根据这个对应表和前一阶段的产物(见表 9-3),我们可以进一步推导出要测试 HRMIS 所需的测试类型,针对“增加员工信息”的测试我们需要进行功能测试、完整性测试、用户界面测试(见表 9-5)。

表 9-5　HRMIS 的测试类型分析实例

质量特性对应表				
原始需求描述	标识	测试要点	质量特性	测试类型
一条完整的培训信息包括培训的主题、证书、内容、起止时间、费用、地点、机构,其中培训的主题、内容、起止时间、费用、机构为必填项。培训的起始时间不能晚于截止时间,培训费用精确到元角分。每一个输入项的数据规格在数据字典中可以得到	1	输入符合字典要求的各信息后执行保存,检查保存是否成功	功能性/适合性	功能测试
	2	检查每个输入项的数据长度是否遵循数据字典的要求	功能性/适合性、可靠性/容错性	功能测试、完整性测试
	3	检查每个输入项的数据类型是否遵循数据字典的要求	功能性/适合性、可靠性/容错性	功能测试、完整性测试
	4	检查“培训费用”是否满足规定的精度要求	功能性/准确性	功能测试
	5	检查在培训的起止时间早晚于截止时间时,所增加的记录是否保存成功	功能性/适合性	功能测试
	6	检查“培训主题”、“培训内容”、“起止时间”、“培训费用”、“培训机构”是否为必填项	功能性/适合性	功能测试
	7	验证系统对数据重复的检查	功能性/适合性	功能测试
	8	针对页面中文字、表单、图片、表格等元素,检查每个页面各元素的位置是否协调,各元素的颜色是否协调,各元素的大小比例是协调	易用性/易操作性	用户界面测试
	9	页面信息内容显示是否完整	易用性/易操作性	用户界面测试
	10	检查是否有功能标识,功能标识是否准确、清晰	易用性/易理解性	用户界面测试、功能测试
	11	最大化、最小化、还原、切换、移动窗口时是否能正常的显示页面	易用性/易操作性	用户界面测试

另外,为了避免遗漏,在确定测试类型时,通常还需要考虑以下几个方面:

- 测试计划中确定的测试对象、测试项、要测试的特征包括的内容;
- 软件文档中是否包含测试类型相对应的情况的说明;
- 列出的常见测试类型是否已完全覆盖了被测软件;
- 被测软件的某些特殊情况是否已包含在所列出的测试类型中。

3. 测试需求跟踪矩阵

在实际的项目中,期望在项目开始阶段就获得稳定的需求是不现实的,软件需求在整个开发过程中处于不断调整和完善的状态,即使到了系统测试阶段,软件需求也存在着变

化的风险。软件需求的频繁变更增加了软件测试工作的复杂性,直接影响测试的质量。

需求变更不可避免,关键在于如何使得需求的变更始终处于受控之下,这就需要对需求变更进行有效地管理。对于测试工作而言,需求变更管理是一个相对被动的过程,软件需求发生了变更,测试需求必须随之变化,这就要求对需求变更做出快速的反应。在软件测试中,可以采用测试需求跟踪矩阵的方式对需求变更实施管理。首先,建立软件需求与测试需求的跟踪关系表;然后,将软件需求与软件测试需求对应起来,形成完整的软件需求与软件测试需求的跟踪关系矩阵。

在 HRMIS 的测试中,根据前面几节的工作成果,可以得到 HRMIS 的测试需求跟踪矩阵表(见表 9-6)。

表 9-6 HRMIS 的测试需求跟踪矩阵

软件需求			测试需求		
软件需求标识		软件需求描述	测试需求标识	测试要点	测试类型
3.6.1 培训信息维护	增加培训信息	一条完整的培训信息包括培训的主题、证书、内容、起止时间、费用、地点、机构,其中培训的主题、内容、起止时间、费用、机构为必填项。培训的起始时间不能晚于截止时间,培训费用精确到元角分。每一个输入项的数据规格应遵循数据字典的要求	1	输入符合字典要求的各信息后执行保存,检查保存是否成功	功能测试
			2	检查每个输入项的数据长度是否遵循数据字典的要求	功能测试、完整性测试
			3	检查每个输入项的数据类型是否遵循数据字典的要求	功能测试、完整性测试
			4	检查"培训费用"是否满足规定的精度要求	功能测试
			5	检查在培训的起止时间早晚于截止时间时,所增加的记录是否保存成功	功能测试
			6	检查"培训主题"、"培训内容"、"起止时间"、"培训费用"、"培训机构"是否为必填项	功能测试
			7	验证系统对数据重复的检查	功能测试
			8	针对页面中文字、表单、图片、表格等元素,检查每个页面各元素的位置是否协调,各元素的颜色是否协调,各元素的大小比例是协调	用户界面测试
			9	页面信息内容显示是否完整	用户界面测试
			10	检查是否有功能标识,功能标识是否准确、清晰	用户界面测试、功能测试
			11	最大化、最小化、还原、切换、移动窗口时是否能正常的显示页面	用户界面测试

测试需求跟踪矩阵是测试需求分析阶段的最终产物,在整个后续的测试工作中,需求

跟踪关系需要不断的维护,维护工作体现在两个方面。一方面,软件需求一旦发生变化,就要对需求跟踪表进行维护,启动配置管理过程,将与软件需求变更相关的内容进行同步变更;另一方面,随着测试工作的进行,会不断添加新的跟踪内容,需要对跟踪表进行扩展。例如,测试设计阶段的测试用例、测试执行阶段的测试记录和测试缺陷都可以添加到跟踪矩阵中。

举例:表 9-7 给出了软件需求、测试需求与测试用例三者之间的跟踪表的格式。

表 9-7　跟踪表格式

软件需求		测试需求			测试用例	
软件需求标识	软件需求描述	测试需求标识	测试要点	测试类型	用例标识	用例描述

9.2.3　测试需求评审

形成测试需求矩阵后,需要对测试需求进行评审。评审的主要内容是测试需求的完整性和准确性。测试需求完整性审查应保证测试需求能充分覆盖软件需求的各种特征,重点关注功能要求、数据定义、接口定义、性能要求、安全性要求、可靠性要求、系统约束等方面,同时还应关注是否覆盖开发人员遗漏的、系统隐含的需求;测试需求准确性审查应保证所描述的内容能够得到相关各方的一致理解,各项测试需求之间没有矛盾和冲突,各项测试需求在详尽程度上保持一致,每一项测试需求都可以作为测试用例设计的依据。

测试需求评审可以归为静态测试的范畴,通常通过正式的评审会议来进行。在正式评审小组中,一般存在多种角色,包括协调人、作者、评审员等。为了保证评审的质量和效率,需要精心挑选评审员,保证不同类型的人员都要参与进来。评审员通常包括开发经理、项目经理、测试经理、系统分析人员、相关开发人员和测试人员等。在不同类型的人员中,应选择那些对系统有足够了解的人员参与进来,否则很可能会降低评审的效率。

在评审过程中,一般来说,作者应事先主动的向评审员介绍测试需求分析的背景、过程、思路,说明相关的细节,征集大家的意见。在介绍前应先把文档发给评审员。评审员在获得文档后应仔细阅读,将阅读中发现的问题、不明白的地方一一记下来,通过邮件发给文档的作者,或通过其他形式(面对面会谈、电话、远程会议等)进行交流。通过交流,大家达成一致的认识和理解,并修改不正确、不清楚的地方。

如果通过一些非正式的形式(如相互评审、走查等),不能很好地完成测试需求的评审,就必须通过正式形式(如小组评审)来完成这一工作。对于比较大型的项目来说,一次评审会议也许还不够,还要通过多次的评审会议才能最后达成一致。

测试需求评审的形式,一般包括以下几种,在使用时可以结合实际情况,灵活应用。

- 相互评审、交叉评审:甲和乙在一个项目组,处在一个领域,但工作内容不同,甲的工作成果交给乙审查,乙的工作成果交给甲审查。相互评审是最不正式的一种评审形式,但应用方便、有效。

- 轮查：又称分配审查方法，是一种异步评审方式。作者将需要评审的内容发送给各位评审员，并收集他们的反馈意见。
- 走查：作者将测试需求在现场向一组同事介绍，以收集大家的意见。希望参与评审的其他同事可以发现其中的错误，并能进行现场讨论。这种形式介于正式和非正式之间。
- 小组评审：通过正式的小组会议完成评审工作，是有计划的和结构化的评审方式。评审定义了评审会议中的各种角色和相应的责任，所有参与者在评审会议的前几天就拿到了评审材料，并对该材料进行了独立研究。
- 审查：审查和小组评审很相似，但更为严格，是最系统化、最严密的评审形式，包含了制定计划、准备和组织会议、跟踪和分析审查结果等。

测试需求在评审后，测试分析人员应根据评审意见做出修改，然后继续评审，直至通过评审。评审最后应达到以下结果：所有参与方达成一致；已发现的问题被阐述清楚、被修正。

本章小结

测试需求是测试设计和开发测试用例的基础，是解决“测什么”的问题，测试需求分析是对软件需求进行测试分析，并对分析结果进行合理的测试分解，逐步规约到测试类型的过程。测试需求分析可以划分为需求采集、需求分析、需求评审三个过程。其中，需求采集是提取需求来源中的那些具有可测试性的需求或特性，形成原始测试需求表；需求分析是对原始测试需求进行测试分析，得到测试要点，然后针对这些测试要点，参考质量特性定义，得出软件的质量特性需求，再从测试类型出发，确定出软件需要进行的测试类型，最后得到测试需求跟踪矩阵；需求评审是对测试需求进行的完整性和准确性审查，以获得各方的认可。

CHAPTER

第 10 章 测试方法与测试设计

软件测试的重要性是毋庸置疑的，但是测试的投入却不是无限制增加的，那么如何才能以最少的资源投入，在最短的时间内完成测试，同时又能尽可能多的发现软件缺陷，提升软件的质量呢？这看似悖论的一个问题，成了众多软件公司探索和追求的目标。这个问题的核心实际上是测试方法的问题，做任何事情都要讲究方法，软件测试之好坏最终也要依赖于此。一个好的测试方法能够使测试工作事半功倍，而一个不合适的方法往往会把软件测试引向错误的途径，造成测试资源的浪费。

本章将从静态与动态两个角度全面介绍软件测试中的白盒和黑盒测试技术，讲解软件测试用例的设计、开发，并以 HRMIS 的测试为例从单元测试、集成测试、系统测试和验收测试等四个递进的测试阶段全面揭示软件测试设计的过程和方法。

10.1 静态测试

静态测试是指不实际运行被测程序，采用人工检测和计算机辅助静态分析的手段，对各类项目文档和程序源代码进行分析与检查。因此，静态测试可以分为文档的静态测试和代码的静态测试两部分，对文档的静态测试方法主要是以检查单的形式进行，而对代码的静态测试方法一般采用代码审查、代码走查和静态分析。

10.1.1 文档检查/审查

静态测试可以用于对各种软件文档进行测试，是软件开发中十分有效的质量控制方法之一。对文档的静态测试一般采用评审会的方式进行评审，并以检查单的方法来检查文档。

评审会的过程分三个阶段：

① 由软件的软件开发单位负责人、用户代表、开发小组成员、标准人员等组成评审小组，必要时可以邀请外单位的专家参加。

② 开会前，由开发单位负责人确定评审的具体内容，并将评审材料发给评审小组成员，要求做好评审准备。

③ 由开发单位负责人主持评审会，根据文档编制者对该文档的说明和检查单，由评审小组成员进行评议、评审，评审结束作出评审结论。

检查单是一些在审阅过程中经常会问的问题清单，评审人员根据这些问题对照检查文档。针对不同的软件中间产品，静态测试的内容也不尽相同，对不同的文档进行静态测试的内容可以体现在对特定文档的检查单中。下面以需求说明书、设计说明书为代表，列举一些检查单，在实践中可根据实际工作情况进行增减。

1. 需求说明书检查单

对需求说明书的测试着重于审查用户需求描述及其解释是否完整、准确。下面给出对需求说明书进行静态测试的检查单。

1）正确性

① 需求定义是否满足软件标准、规范的要求？

② 算法和规则是否有科技文献或其他文献作为基础？

③ 有哪些证据说明用户提供的规则或规定是正确的？

④ 是否正确的定义了各种故障模式和错误类型所需的反应？

⑤ 是否所有的功能都有明确的目的？

⑥ 是否存在对用户无意义的功能？

⑦ 每个需求定义是否都合理，经得起推敲？

⑧ 对设计和实现的限制是否都有论证？

2）易理解性

① 每一个需求是否只有一种解释？

② 语言是否有歧义？

③ 是否使用了形式化或半形式化的语言？

④ 是否包含了实现的细节，是否过分细致了？

⑤ 需求定义是否足够清楚和明确，使其已能作为开发设计说明书的基础？

⑥ 是否有术语定义一览表？

⑦ 功能性需求的描述结构化、流程化是否良好？

3）完备性

① 是否包含了有关功能、性能、限制、目标、质量等方面的所有需求？

② 功能性需求是否覆盖了所有非正常的处理？

③ 是否有漏掉的功能？

④ 是否有漏掉的输入、输出或条件？

⑤ 是否定义与说明了系统输入、输出的类型、值域、单位、格式、精度？

⑥ 是否考虑了关于人机界面的需求？

⑦ 是否对所有与时间有关的功能方面都做了考虑？

⑧ 是否对各种操作模式下的环境条件做了规定？

⑨ 是否识别并定义了将来可能会变化的需求？

⑩ 是否包含了有关文件（如质量手册、配置计划）中所规定的特定需求？

4）一致性

① 各需求之间是否一致，是否有冲突和矛盾？

② 是否使用了标准术语和定义形式？

③ 所规定的模型、算法、数据格式是否相容？

④ 是否说明了软件环境对软件的影响？

⑤ 是否说明了软件对其系统和环境的影响？

5）可行性

① 定义的功能是否能通过现有技术实现？

② 是否能够达到有关质量的要求？

③ 所有的功能是否能在相应的限制条件下实现？

④ 所规定的模型、算法是否能解决需求中存在的问题？

6）可验证性

① 所定义的功能正确性是否可以判断？

② 系统的非功能性需求是否有验证的标准和方法？

③ 数学函数的定义是否使用了精确定义的语法和语义符号？

7）可修改性

① 对需求定义的描述是否易于修改？

② 是否采用了良好的文档结构？

③ 是否有冗余的信息？

8）可追踪性

① 每一项需求定义是否可以确定其来源？

② 功能的限制条件是否可以找到其存在的理由？

③ 需求定义是否便于向后继开发阶段查找信息？

2. 设计说明书的检查单

对设计说明书的测试着重于分析设计是否与需求说明一致，所采用的算法是否适于待解决的问题，需求是否都被满足等。下面给出对设计说明书进行静态测试的检查单。

1）正确性

① 设计文档是否满足有关标准的要求？

② 设计中若包含额外的功能，这些功能的必要性是否经过论证？

③ 程序是否会完成所需的功能？

④ 是否与所描述的操作环境一致？

⑤ 界面设计是否与文档所描述的界面部分一致？

2）易理解性

① 是否避免了不必要的成分和表达形式？

② 是否造成歧义性解释？

3）完备性

① 需求定义中的需求是否都已完成？

② 是否有充分的数据来保证设计的完整性？

③ 算法、公式等是否充分、准确？

④ 是否明确说明了产品开发、测试中所需要的软硬件？

⑤ 是否标识出了每一个输入、输出和数据库成分，其描述是否详细到了可以编码的程度？

⑥ 是否包含了所有的处理步骤？

⑦ 是否考虑了所有的可能情况和条件？

⑧ 是否参照了有关的设计标准？

⑨ 是否已记录设计时的权衡考虑，该文件是否包括了权衡选择的标准和不选择其他方案的原因？

4）一致性

① 是否使用标准的术语和定义？

② 文档风格和详细程度是否前后一致？

③ 设计是否包含内在矛盾？

④ 输入输出的格式是否一致？

⑤ 界面之间是否相容？

⑥ 类似的功能和相关的功能设计是否一致？

⑦ 计算中的计量单位和计算精度是否一致？

5）可行性

① 所设计的模型和算法对于应用领域来说是否可以接受？

② 在可用资源下，所设计的功能是否能实现？

③ 是否存在错误的、缺少的或不完整的逻辑？

④ 设计能否在所规定的限制条件下和开发代价下实现？

6）可验证性

① 每一个功能的描述是否使用了良好的术语和符号？

② 是否可以验证每个功能与需求定义相一致？

③ 是否定量的说明了使用条件、限制等内容？

④ 是否可以产生测试数据？

7）可修改性

① 每一个子程序是否都只实现一个功能？

② 是否使用了信息隐藏技术？

8）可追踪性

① 是否包含了设计与需求说明中的需求、设计限制等内容的对应关系？

② 是否对所有功能进行了适当的标识？

③ 是否包含修改历史的记录，所有对设计修改和修改理由是否记录在案并赋予编号？

④ 是否标识出设计中所包含的需求定义之外的功能？

9）结构化

是否使用了层次式的控制结构？

10）模块性

① 是否采用了模块化的机制？

② 是否使系统由一系列相对较小的、以层次结构相互联系的子程序组成？

③ 是否每一个子程序只完成一个特定的功能？

④ 是否使用了特殊的规则来限制子程序的大小？

10.1.2 代码检查/审查

对代码的静态测试方法一般采用代码审查、代码走查和静态分析，静态分析一般包括控制流分析、数据流分析、接口分析和表达式分析。

1. 代码审查

代码审查主要检查代码和设计的一致性，代码执行标准的情况，代码逻辑表达的正确性，代码结构的合理性，代码的可读性等。

代码审查由若干程序员和测试员组成审查小组，一般由四人以上组成，分别为组长、资深程序员、程序编写者、测试人员。组长不能是被测试程序的编写者，组长主要负责分配资料、安排计划、主持开会、记录并保存被发现的差错。

代码审查的过程分为四个阶段：

① 准备阶段。组长把设计规格说明书、控制流程图、程序文本及有关要求、规范等材料分发给小组成员，作为审查的依据。被测程序的设计人员、编码人员向审查组详细说明有关材料，并回答审查组成员所提出的有关问题。

② 程序阅读。审查组人员仔细阅读代码和相关材料，对照代码审查单，记录问题及明显缺陷。

③ 会议审查。组长主持会议，程序员逐句讲解程序的逻辑，其他人员提出问题，利用代码审查单进行分析讨论，对讨论的各个问题形成结论性意见。

④ 形成报告。会后将发现的差错形成代码审查问题表，并交给程序开发人员。对发现差错较多或发现重大差错的，在改正差错之后，再次进行会议审查。

代码审查单是一份常见错误的清单，它把程序中可能发生的各种错误进行分类，对每一类列举出尽可能多的典型错误，供与会者对照检查。以下是一个推荐的代码审查单，可以根据实际工作经验和具体被测程序对以下内容进行增减：

（1）寄存器使用(仅限定在机器指令和汇编语言时考虑)。

- 如果需要一个专用寄存器，指定了吗？
- 宏扩展或子程序调用是否使用了已使用着的寄存器而没有保存数据？
- 默认使用的寄存器的值正确吗？

（2）格式。

- 嵌套的 IF 是否已正确地缩进？
- 注释是否准确并有意义？
- 是否使用了有意义的标号？
- 代码是否基本上与开始时的模块模式一致？

- 是否遵循全套的编程标准？

(3) 入口和出口的连接。

- 初始入口和最终出口是否正确？
- 对另一模块的每一次调用，全部所需的参数是否已传送给每一个被调用的模块？被传送的参数值是否正确的设置？对关键的被调用模块的意外情况(如丢失、混乱)有处理吗？

(4) 程序语言的使用。

- 是否使用一个或一组最佳动词？
- 模块中是否使用完整定义的语言的有限子集？
- 是否使用了适当的跳转语句？

(5) 存储器使用。

- 每一个域在第一次使用前正确地初始化了吗？
- 规定的域正确吗？
- 每个域是否有正确的变量类型声明？
- 存储器重复使用吗？可能产生冲突吗？

(6) 测试和转移。

- 是否进行了浮点相等比较？
- 测试条件是否正确？
- 用于测试的变量是否正确？
- 每个转换目标正确并至少执行一次？
- 三种情况(大于 0，小于 0，等于 0)是否已全部测试？

(7) 性能。

- 逻辑是否被最佳地编码？
- 提供的是一般的出错处理还是异常的例程？

(8) 可维护性。

- 所提供的列表控制是否有利于提高可读性？
- 标号和子程序符合代码的逻辑意义吗？

(9) 逻辑。

- 全部设计是否均已实现？
- 编码是否做了设计所规定的内容？
- 每个循环是否执行了正确的次数？
- 输入参数的所有异常值是否已直接测试？

(10) 可靠性。

- 对从外部接口采集的数据有确认吗？
- 遵循可靠性编程要求了吗？
- 是否存在内存泄露的问题？

(11) 软件多余物。

- 是否有不可能执行到的代码？

- 是否有既使不执行也不影响程序功能的指令？
- 是否有多余的程序单元？
- 是否有未引用的变量、标号、常量？

代码审查问题表应写明所查出的差错类型、差错类别、差错严重程度、差错位置、差错原因。差错类型包括文档差错、编程语言差错、逻辑差错、接口差错、数据使用差错、编程风格不当、软件多余物。差错类别有遗漏、错误、多余。

代码审查是一种多人一起进行的测试活动，要求每个人尽量多提出问题，展开讨论。同时讲述程序者在讲解过程中也会突然发现许多原来自己没有发现的问题，这时要放慢讲解进度，把问题分析出来。

2．代码走查

代码走查与代码审查基本相同。

代码走查一般由四人以上组成，分别为组长、秘书、资深程序员、测试人员。被测试程序的编写者可以作为走查组成员。组长负责分配资料、安排计划、主持开会，秘书记录被发现的差错。

代码走查的过程分为四个阶段：

① 准备阶段。组长把有关材料分发给走查小组每个成员，走查组详细阅读材料和认真研究程序。

② 生成实例。测试人员为所测程序准备一些有代表性的测试实例。

③ 会议走查。组长主持会议，其他人员充当计算机，对测试实例用头脑来执行程序，也就是让测试实例沿程序的逻辑运行一遍，并由测试人员讲述程序执行过程，把程序的状态（如变量的值）记录在纸上或黑板上以供监视，秘书记录下发现的问题。

④ 形成报告。会后将发现的差错形成报告，并交给程序开发人员。对发现差错较多或发现重大差错的，在改正差错之后再次进行会议走查。

代码走查是一种多人一起进行的测试活动，要求测试人员尽量多提供测试实例，这些测试实例是作为怀疑程序逻辑与计算差错的启发点，在随测试实例游历程序逻辑时，在怀疑程序的过程中发现差错。这种方法不如代码审查检查的范围广，差错覆盖全。

3．静态分析

静态分析的目的是发现软件源代码和软件模型中的缺陷，一般包括控制流分析、数据流分析、接口分析、表达式分析等。静态分析是在程序编译通过之后，其他静态测试之前进行的。

静态分析可以完成下述工作：

① 提供间接涉及程序缺陷的信息：

- 每一类型语句出现的次数；
- 所有变量、常量的交叉引用表；
- 标识符的使用方式；
- 过程的调用层次；
- 违背编码规则；

- 程序结构图、程序流程图；
- 子程序规模、调用/被调用关系、扇入/扇出数。

② 进行语法/语义分析，提出语义或结构要点，供进一步分析。

③ 进行符号求值。

④ 为动态测试选择测试用例进行预处理。

静态分析常需要使用软件工具进行。静态分析的对象是计算机程序，程序设计语言不同，相应的静态分析工具也就不同。目前具备静态分析功能的软件测试工具有很多，如 Rational 公司的 Purify、Telelogic 公司的 Logiscope、Macabe 公司的 Macabe、PR 公司的 PRQA 等。

1）控制流分析

控制流分析是使用控制流程图系统地检查被测程序的控制结构的工作。控制流按照结构化程序规则和程序结构的基本要求进行程序结构检查。这些要求是被测程序不应包含：

① 转向并不存在的语句标号。

② 没有使用的语句标号。

③ 没有使用的子程序定义。

④ 调用并不存在的子程序。

⑤ 从程序入口进入后无法达到的语句。

⑥ 不能达到停止语句的语句。

控制流程图是一种简化的程序流程图，控制流程图由“结点”和“弧”两种图形符号构成。

2）数据流分析

数据流分析是用控制流程图来分析数据发生的异常情况，这些异常包括被初始化、被赋值或被引用过程中行为序列的异常。数据流分析也作为数据流测试的预处理过程。

数据流分析首先建立控制流程图，然后在控制流程图中标注某个数据对象的操作序列，遍历控制流程图，形成这个数据对象的数据流模型，并给出这个数据对象的初始状态，利用数据流异常状态图分析数据对象可能的异常。

数据流分析可以查出引用未定义变量、对以前未使用的变量再次赋值等程序差错或异常情况。

3）接口分析

接口分析主要用在程序静态分析和设计分析。接口一致性的设计分析涉及模块之间接口的一致性，以及模块与外部数据库之间的一致性。

程序的接口分析涉及子程序以及函数之间的接口一致性，包括检查形参与实参的类型、数量、维数、顺序、使用的一致性，检查全局变量和公共数据区在使用上的一致性。

4）表达式分析

对表达式进行分析，以发现和纠正在表达式中出现的错误。表达式错误主要有以下几种：

① 括号使用不正确。

② 数组引用错误。

③ 作为除数的变量可能为零。

④ 作为开平方的变量可能为负。

⑤ 作为正切值的变量可能为 $\pi/2$。

⑥ 浮点数变量比较时产生的错误。

10.2 动态测试

动态测试是在计算机上运行被测的程序代码或代码段，通过输入测试用例对其运行情况(输入/输出的对应关系)进行观察，并对测试结果进行分析。动态测试是建立在对程序的执行过程中，根据是否对被测对象内部的了解，可以分为黑盒测试和白盒测试(前面所述的代码审查、代码走查和静态分析本质上属于白盒测试的范畴)。

黑盒测试又称功能测试、数据驱动测试或基于规格说明的测试，这种测试不必了解被测对象的内部情况，而依靠需求规格说明中的功能来设计测试用例。白盒测试又称结构测试、逻辑测试或基于程序的测试，这种测试应了解程序的内部构造，并且根据内部构造设计测试用例。

在软件动态测试过程中，应采用适当的测试方法，实现测试目标。系统测试一般采用黑盒测试方法；集成测试一般主要采用黑盒测试方法，辅之以白盒测试方法；单元测试一般采用白盒测试方法，辅之以黑盒测试方法。

10.2.1 测试用例概述

1. 什么是测试用例

软件测试的重要性是毋庸置疑的，但是如何以最少的资源投入，用最短的时间出色地完成测试，尽可能多地发现软件系统的缺陷，以提升软件的质量是一个恒久不变的课题。这个课题本质上就是如何为被测系统建立一套优秀的测试方案。影响软件测试的因素很多，例如需求定义的精确程度、软件本身的复杂度、系统设计的合理性以及开发人员的素质等，这些是开发层面的，那么对于软件测试自身来说不同的测试方法和技术的运用也会导致不一样的测试效果。如何才能保障软件测试质量的稳定呢？在现阶段一个有效的方法就是基于测试用例的测试。

测试用例(Test Case)简单来说就是预先编制的一组系统操作步骤和输入数据、执行条件以及预期结果，用以验证某个程序是否满足某个特定需求的文字。在 GB/T 11457—2006《信息技术 软件工程术语》中，测试用例是这样定义的：

① 为具体的目标(例如，为练习具体的程序路径或验证对特定需求的遵循性)而开发的一组测试输入、执行条件和预料结果。

② 对于测试，规定输入、预料的结果和一组执行条件的文档。

使用测试用例至少可以带来以下好处：

① 在开始实施测试之前设计好测试用例，可以避免盲目测试并提高测试效率。

② 测试用例的使用令软件测试的实施重点突出、目的明确。

③ 使回归测试(有关回归测试的内容我们会在后面的内容中讲解)易于控制,比如在软件版本更新后只需要修改少部分的测试用例便可展开测试工作,降低工作强度,缩短项目周期。

④ 功能模块的通用化和复用化使软件易于开发,而相对于功能模块的测试用例的通用化和复用化则会使软件测试易于开展,并随着测试用例的不断精化其效率也不断攀升。

⑤ 为测试脚本的编制提供依据和标准,是脚本开发的设计规格说明书。

⑥ 为测试度量提供一个更加具有可操作性的维度,例如可以统计用例执行率、用例通过率等来对软件测试过程进行评估。

2. 测试用例的层次模型

测试用例来源于需求,通过需求追溯最终用户的使用场景,根据用户的使用场景构造出测试用例(见图 10-1)。一个用户场景可能会衍生多个测试用例。

测试用例包含两个层面的信息,一个是测试步骤,用来指导测试执行人员遵循什么样的顺序操作被测系统(SUT),可以使即便是对系统不了解的人员也可以顺利的开始测试;另一个是测试数据,测试数据为测试人员提供输入数据,这些数据一般是通过严格的测试用例设计方法(如边界值、等价类等)精选出来的,每一组测试数据代表了一个检查点。

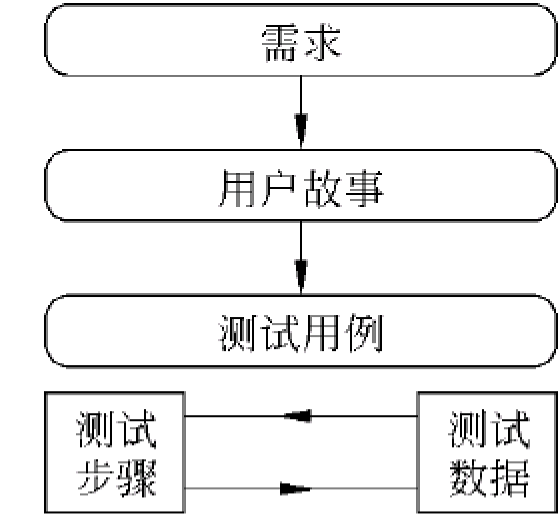


图 10-1 测试用例层次模型示意

3. 测试用例要素

测试用例的编写格式不尽相同,但是测试用例中的一些核心要素不论格式怎么调整也不会丢失,如测试用例的标识、预置条件、测试步骤、预期测试结果等。

本书中即将介绍的测试用例编写要素主要依据 GB/T 15532—2008《计算机软件测试规范》的有关要求,作为一个最新版本的有关软件测试的规范,应具有一定的代表性。

在 GB/T 15532—2008《计算机软件测试规范》中,测试用例包含以下要素:

① 名称和标识。每个测试用例应有唯一的名称和标识符。

注意: 测试用例编号一般可由字母和数字组合而成,用例编号要保持唯一性,易于识别和检索。一个有效的实践是这样的:产品/项目编号-ST-系统测试项名-系统测试子项名-流水编号,通过这种规范的编码规则,可以直接通过用例编号知道是什么产品(或者项目)做的什么类型的测试,测试的对象是什么等重要的识别信息。

② 测试追踪。说明测试所依据的内容来源,如系统测试依据的是用户需求,配置项测试依据的是软件需求,集成测试和单元测试依据的是软件设计(概要设计说明和详细设计说明)。

③ 用例说明。简要描述测试的对象、目的和所采用的测试方法;

注意: 用例说明顾名思义是对测试用例的简单描述。既然要求简单就尽量用概括的语言来描述该测试用例的测试点,尽量采用“什么条件下,做什么”这样的描述语。测试用

例说明要保持唯一性，这对于一些使用自动化管理平台的企业是重要的。

④ 测试的初始化要求。应考虑下述初始化要求：

- 硬件配置：被测系统的硬件配置情况，包括硬件条件或电气状态。
- 软件配置：被测系统的软件配置情况，包括测试的初始条件。
- 测试配置：测试系统的配置情况，如用于测试的模拟系统和测试工具等的配置情况。
- 参数设置：测试开始前的设置，如标志、第一断点、指针、控制参数和初始化数据等的设置。
- 其他对于测试用例的特殊说明。

注意：这个等同于业界通常所说的“预置条件”。

⑤ 测试的输入。在测试用例执行中发送给被测对象的所有测试命令、数据和信号等。对于每个测试用例应提供如下内容：

- 每个测试输入的具体内容(如确定的数值、状态或信号等)及其性质(如有效值、无效值、边界值等)。
- 测试输入的来源(如测试程序产生、磁盘文件、通过网络接收、人工键盘输入等)，以及选择输入所使用的方法(如等价类划分、边界值分析、差错推测、因果图、功能图方法等)。
- 测试输入是真实的还是模拟的。
- 测试输入的时间顺序或事件顺序。

⑥ 期望的测试结果。说明测试用例执行中由被测软件所产生期望的测试结果，即经过验证，认为正确的结果。必要时，应提供中间的期望结果。期望测试结果应该有具体内容，如确定的数值、状态或信号等，不应是不确切的概念或笼统的描述；

⑦ 评价测试结果的准则。判断测试用例执行中产生的中间和最后结果是否正确的准则。对于每个测试结果，应根据不同情况提供如下信息：

- 实际测试结果所需的精度。
- 实际测试结果与期望结果之间的差异允许的上限、下限。
- 时间的最大和最小间隔，或事件数目的最大和最小值。
- 实际测试结果不确定时，再测试的条件。
- 与产生测试结果有关的出错处理。
- 上面没有提及的其他准则。

⑧ 操作过程。实施测试用例的执行步骤。把测试的操作过程定义为一系列按照执行顺序排列的相对独立的步骤，对于每个操作应提供：

- 每一步所需的测试操作动作、测试程序的输入、设备操作等。
- 每一步期望的测试结果。
- 每一步的评价准则。
- 程序终止伴随的动作或差错指示。
- 获取和分析实际测试结果的过程。

⑨ 前提和约束。在测试用例说明中施加的所有前提条件和约束条件，如果有特别限

制、参数偏差或异常处理，应该标识出来，并要说明它们对测试用例的影响；

⑩ 测试终止条件。说明测试正常终止和异常终止的条件。

按照这个测试用例编写要求，测试用例的编写模板如下（GB/T 15532—2008《计算机软件测试规范》）：

<table>
<tr><td colspan="2">用例名称</td><td colspan="3"></td><td colspan="2">用例标识</td><td></td></tr>
<tr><td colspan="2">测试追踪</td><td colspan="6"></td></tr>
<tr><td>用例说明</td><td colspan="7"></td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="6"></td></tr>
<tr><td>软件配置</td><td colspan="6"></td></tr>
<tr><td>测试配置</td><td colspan="6"></td></tr>
<tr><td>参数设置</td><td colspan="6"></td></tr>
<tr><td colspan="8">操作过程</td></tr>
<tr><td>序号</td><td colspan="2">输入及操作说明</td><td>期望的测试结果</td><td colspan="2">评价标准</td><td colspan="2">备注</td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td colspan="3">前提和约束</td><td colspan="5"></td></tr>
<tr><td colspan="3">过程终止条件</td><td colspan="5"></td></tr>
<tr><td colspan="3">结果评价标准</td><td colspan="5"></td></tr>
<tr><td colspan="3">设计人员</td><td></td><td colspan="2">设计日期</td><td colspan="2"></td></tr>
</table>

上面以 GB/T 15532 为基准介绍了测试用例的编写要素，在业界，关于测试用例的编写讨论也颇多，两相比较总体差异不大，其中“重要程度”这个要素在这个规范中没有提及，这个还是比较有意义的，作为一个补充在此介绍一下。

测试用例的重要程度表示了测试用例的优先级，在资源紧张的情况下为我们的取舍提供了依据。重要程度一般分为高中低三个级别：

- 高：保证系统基本功能、重要特性、实际使用频率比较高的用例。
- 中：重要程度介于高和低之间的测试用例。
- 低：实际使用频率不高、对系统业务功能影响不大的模块或功能的测试用例。

4. 测试用例设计原则

设计测试用例时，应遵循以下原则：

- 基于测试需求的原则。应按照测试类别的不同要求，设计测试用例。如，单元测试依据详细设计说明，集成测试依据概要设计说明，系统测试依据用户需求。

- 基于测试方法的原则。应明确所采用的测试用例设计方法。为达到不同的测试充分性要求,应采用相应的测试方法,如等价类划分、边界值分析、猜错法、因果图等方法。
- 兼顾测试充分性和效率的原则。测试用例集应兼顾测试的充分性和测试的效率;每个测试用例的内容也应完整,具有可操作性。
- 测试执行的可再现性原则。应保证测试用例执行的可再现性,即对同样的测试用例,系统的执行结果应当是相同的。

5. 测试用例分类

软件测试的本质是对软件需求规格说明(SRS)的验证,确保软件产品保持一个较高的质量水平,能够满足最终用户的使用要求。在软件开发领域,软件需求是由业务需求到用户需求,从用户需求进一步表述为具体化的软件功能需求这样一个逐渐递进的需求分析过程获得的(如图 10-2 所示)。功能需求构成了 SRS 的主体,也是软件的基本需求。除了功能需求,一般软件要维持一个好的用户体验,还需要对软件的易用性、性能、界面美观性以及软件所应遵从的有关法律法规或其他行业规范等做出要求,所有这些除了功能之外的有关于软件的需求共同构成了软件的非功能需求。在设计测试用例时除了关注软件产品的基本功能之外,对于有关软件产品的非功能需求也应保持足够的关注。对应于功能需求和非功能需求,根据测试用例的用途可以把测试用例做以下归类:

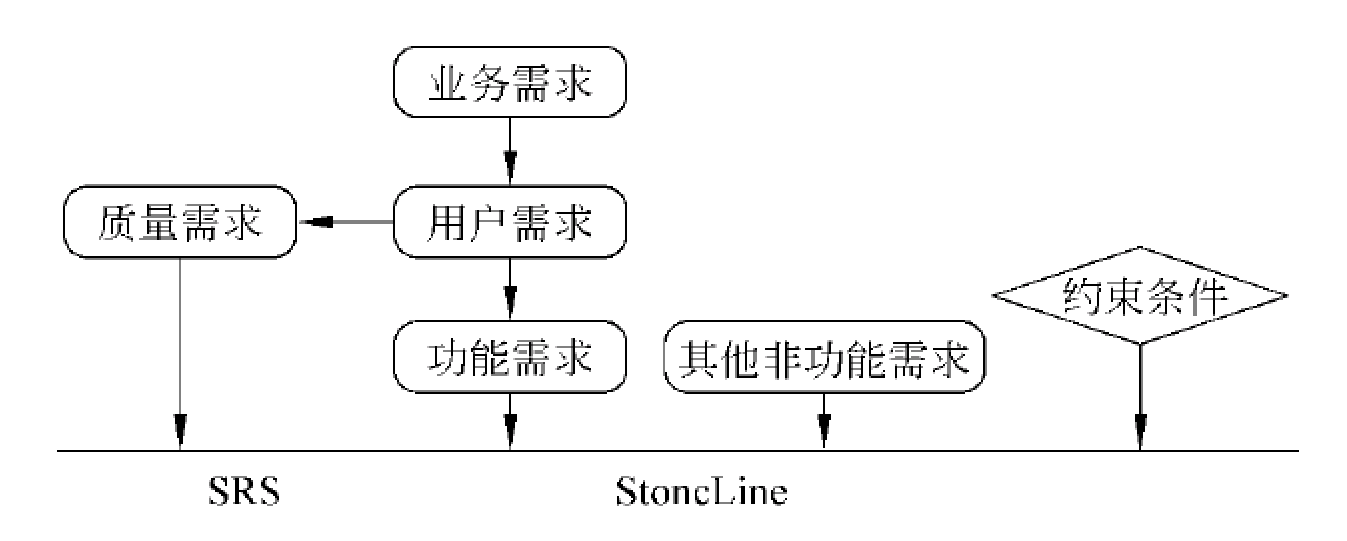


图 10-2 软件需求的层次和构成

1)功能测试用例

功能测试用例一般按照功能模块来组织,系统具有的所有功能都要得到测试,所以针对不同的应用系统,功能测试的内容差异很大。如果把功能测试的内容抽象出来,可以归为正确性、数据、操作、界面等几个方面的测试。

① 正确性。对软件的所有功能执行的正确性与需求的一致性进行检查。

② 数据。对输入、输出的数据项的数据规范进行检查,包括数据类型、长度、精度、格式等,并对有制约关系的数据项进行检查。

③ 操作测试。检查快捷方式、默认值、关键操作、操作方式等方面;并对功能中有操作顺序限制的操作进行验证。

④ 数据接口。对软件中的数据接口与需求的一致性、正确性等进行验证。根据接口的类型,通常可以分为文件接口、服务接口、数据库访问接口三种。

进行功能测试用例的设计,首先要进行功能规格的分析。主要是对各个功能模块中

输入数据的规格和约束进行分析;对每个具体功能的处理机制、操作顺序进行分析;对每个具体功能的输出结果进行分析,在分析时需要考虑数据长度、类型、大小、数据项之间的制约关系等约束,然后结合黑盒测试技术设计测试用例。

2）非功能测试用例

非功能测试用例用来验证系统是否满足非功能测试需求,非功能测试的内容大体可以分为以下几个方面的测试:

① 安全性。对软件中与安全保密性相关的内容进行检查。通常包括传输过程中的安全性、数据存储的安全性、权限控制、登录验证、操作追踪和审计、密码策略等内容。并可以根据软件具体情况,采用模拟攻击等手段(如 SQL 注入、缓冲溢出、拒绝访问等)进行测试。

② 演示功能。对软件中的演示功能的可用性、有效性等进行测试。

③ 程序可定制内容。对软件中的可定制内容的可用性、有效性进行测试。

④ 软件帮助内容。对软件帮助的可用性、有效性等进行测试。

⑤ 系统参数定制。检查利用参数变更软件的有效性和可用性。

⑥ 系统日志。对系统日志的完整、正确、有效性等方面进行检查。

⑦ 软件效率。对软件的响应时间、吞吐量、资源利用等情况进行测试。并可以拓展到强度测试、大数据量等模拟极限情况下的测试。

⑧ 软件重启与修复能力。对系统在出现问题后,能否正确重启和容易地修复问题等情况进行验证。

⑨ 软件安装。对软件进行安装和重新安装。

⑩ 环境兼容。对软件兼容的硬件、软件(包括系统软件和应用软件)、网络环境进行测试。

⑪ 兼容同类软件能力。对兼容同类型软件的情况进行验证,通常包括数据兼容、功能与操作方式的一致等方面。

⑫ 系统可靠性。对软件进行一些破坏性测试,比如网络中断、掉电、机器突然重启等。

⑬ 数据传输可靠性。对软件的传输的可靠性等进行验证,如丢包、续传、错误包、传输内容验证等。

⑭ 用户界面。检查软件以及软件执行过程中的界面、图形、文字、信息和标识是否容易理解、易于浏览、布局合理等。

进行非功能测试用例的设计,应明确软件的各种非功能性需求。有时,软件的文档中可能没有明确说明这些需求,但根据业务的应用环境、软件的实现机理等,隐含地包含了这些需求,因此进行分析设计时,既要归纳出明确提出或说明的软件非功能性需求,还要分析出隐含的非功能性需求。

6. 测试用例评审

测试用例编写完成后,一般要经过评审才能作为正式的测试用例使用。评审一般由业务代表、需求分析人员、设计人员和测试人员共同参与。一般评审工作可以从以下角度考虑:

- 需求的变更部分(新增、修改)是否已经体现在了测试用例中?
- 测试用例是否覆盖了《需求规格说明书》和测试需求?
- 用例编号和需求是否对应?是否填写了《需求跟踪矩阵》?
- 非功能测试需求或不可测试需求是否在用例中列出并说明?
- 用例设计是否至少包含了一个正面和一个反面的用例?
- 每个测试用例是否清楚地填写了测试特性、步骤、预期结果?步骤/输入数据部分是否清晰,是否具备可操作性?测试用例是否包含测试数据、测试数据的生成办法或者输入有无相关描述?每个测试用例是否都阐述预期结果和评估该结果的标准?
- 测试用例使用了什么设计方法?

10.2.2 白盒测试用例设计方法

1. 逻辑覆盖法

逻辑覆盖法是以程序内部的逻辑结构为基础的测试技术,它考虑的是测试数据执行(覆盖)程序的逻辑覆盖程度。使用这一方法要求测试人员对程序的逻辑结构有清楚地了解,甚至要能掌握源程序的所有细节。从覆盖源程序语句详尽程度的不同,逻辑覆盖可以分为:语句覆盖、判定覆盖、条件覆盖、判定-条件覆盖、多条件覆盖。

下面以图 10-3 所示的程序段为例,分别给以说明。对于给出的源程序示例,可以得到程序流程图,可以看出这是一个非常简单的程序,共有两个判断,4 条不同路径。对第一个判定取假分支,对第一个判定取真分支,对第二个判定取假分支,对第二个判定取真分支,分别命名为 b、c、d 和 e。4 条路径表示为 abdf、acdf、abef 和 acef。

示例程序:

```
Dim A,B,X As Integer
If (A>1 AND B=0) Then
  X=X-A
End If
If (A=5 OR X>1) Then
  X=X+ 3
End If
X=B/X
```

1) 语句覆盖

语句覆盖(Statement Coverage,SC)的含义是设计若干个测试用例,使每一个可执行语句至少被执行一次。

在所举的示例中,所有可执行语句都在路径 acef 上,所以选择路径 acef 设计测试用例,就可以覆盖所有的可执行语句。可以构造以下测试用例即可实现:

```
A=5,B=0,X=4(覆盖 acef)
```

从程序中每个可执行语句都得到执行这一点来看，语句覆盖的方法似乎能够比较全面的检验每一个可执行语句，但是语句覆盖对程序执行逻辑覆盖很低，不能准确判断运算中逻辑关系错误。假如，把第一个判定语句中的 AND 错写成 OR，或把第二个判定语句中的 OR 错写成 AND，用上面的测试用例仍可覆盖所有可执行语句，但不能发现其中的逻辑错误。

与后面介绍的其他覆盖相比，语句覆盖是最弱的逻辑覆盖准则。

2）判定覆盖

判定覆盖（Decision Coverage，DC）也被称为分支覆盖，它的含义是设计若干个测试用例，使得程序中的每一个取“真”分支和取“假”分支至少执行一次。

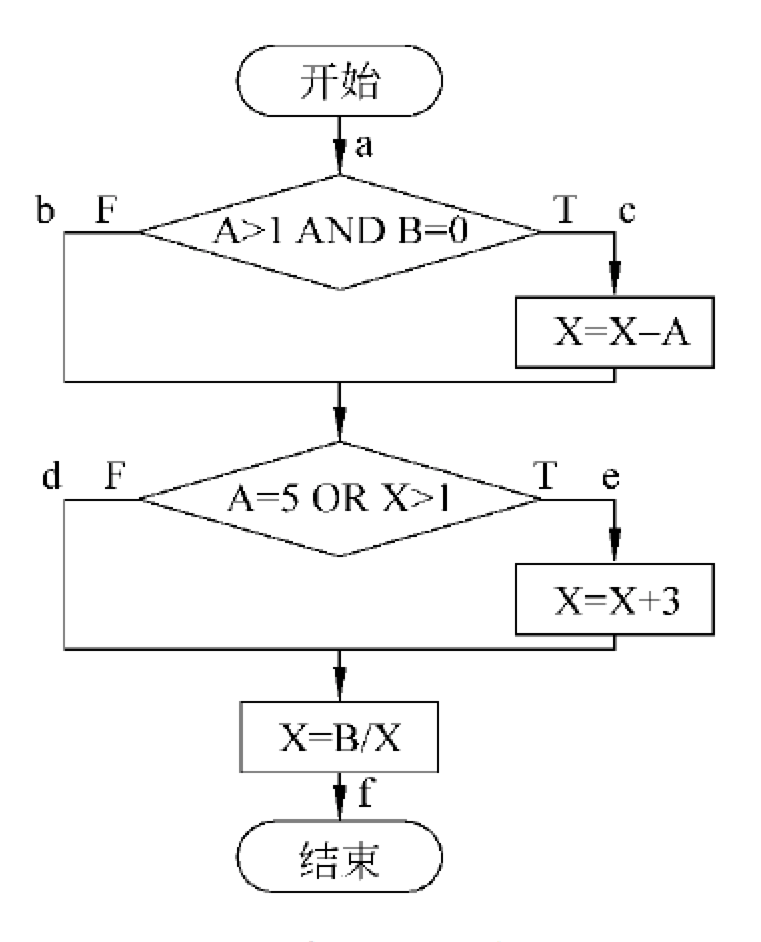


图 10-3 被测程序的流程图

在所举的示例中，选择路径 acef 和 abdf，或 abef 和 acdf，可得到满足要求的测试用例，测试用例可以设计为：

A=5,B=0,X=4(覆盖 acef)
A=1,B=1,X=1(覆盖 abdf)

或

A=5,B=1,X=1(覆盖 abef)
A=3,B=0,X=1(覆盖 acdf)。

判定覆盖比语句覆盖更强一些，通过了每个分支的测试，各语句都被执行了。但也有不足，如上述的测试用例不能发现把第二个判定条件 X>1 错写成 X<1 的错误。所以，判定覆盖还不能保证一定能查出在判定的条件中存在的错误。因此需要更强的逻辑覆盖标准。

3）条件覆盖

条件覆盖(Condition Coverage,CC)的含义是设计若干个测试用例，运行被测程序，使得程序中每一判定语句中每个逻辑条件的可能取值至少执行一次。

按照这一定义，在所举的示例中，对于第一个判定条件：

- 条件 A>1，取真值时为 T1，取假值时为 F1。
- 条件 B=0，取真值时为 T2，取假值时为 F2。

对于第二个判定条件：

- 条件 A=5，取真值时为 T3，取假值时为 F3。
- 条件 X>1，取真值时为 T4，取假值时为 F4。

则测试用例可以设计为：

测试用例	通过路径	条件取值	覆盖分支
A=5,B=1,X=1	abef	T1,F2,T3,F4	b,e
A=1,B=0,X=3	abef	F1,T2,F3,T4	b,e

也可以设计为：

测试用例	通过路径	条件取值	覆盖分支
A=5,B=0,X=4	acef	T1,T2,T3,T4	c,e
A=1,B=0,X=1	abdf	F1,T2,F3,F4	b,d
A=5,B=1,X=1	abef	T1,F2,T3,F4	b,e

两组测试用例都满足了条件覆盖,仔细分析可以发现,第二组在满足条件覆盖的同时,把所有判定的分支也覆盖了,但第一组只覆盖了第一个判定的取假分支和第二个判定的取真分支。为了解决这一矛盾,需要对条件和分支兼顾。

4）判定-条件覆盖

判定-条件覆盖(Decision/Condition Coverage,DCC)是将判定覆盖、条件覆盖结合起来的一种测试用例设计方法。它的含义是设计足够的测试用例,使得判定中每个条件的所有可能取值至少执行一次,同时所有判定的可能结果也至少被执行一次。

对于所举示例,测试用例可以设计为：

测试用例	通过路径	条件取值	覆盖分支
A=5,B=0,X=4	acef	T1,T2,T3,T4	c,e
A=1,B=1,X=1	abdf	F1,F2,F3,F4	b,d

但判定一条件覆盖也有一定的缺陷,从表面上来看,它测试了所有条件的取值,但是事实并非如此,某些条件掩盖了另一些条件。例如对于条件表达式“A>1 AND B=0”来说,若“A>1”的测试结果为假,可以立刻确定表达式的结果为假,这时往往就不再测试“B=0”的取值了,条件“B=0”就没有被检查。同样对于条件表达式“A=5 OR X>1”来说,若“A=5”的测试结果为真,就可以立即确定表达式的结果为真,条件“X>1”就没有被检查。因此,采用判定一条件覆盖,逻辑表达式中的错误不一定能够查得出来。此外,也不能保证覆盖所有的路径,如本例中只覆盖了 acef、abdf,没有覆盖 abef、acdf。

5）条件组合覆盖

条件组合覆盖也称多条件覆盖(Multiple Condition Coverage,MCC),它的含义是设计足够的测试用例,使得每个判定中条件的各种组合都至少被执行一次。显然,满足条件组合覆盖的测试用例一定满足判定覆盖、条件覆盖和判定-条件覆盖的。

对于所举示例,判定语句中四个逻辑条件,每个逻辑条件有两种取值,因此共有 $2^3=8$ 种可能组合。先对各个判定的条件取值组合加以标记,如表 10-1 所示。

表 10-1　判定条件取值

组合编号	条件取值的组合	标　　记	说　　明
(1)	A>1,B=0	T1,T2	属第一个判定的取真分支
(2)	A>1,B≠0	T1,F2	属第一个判定的取假分支
(3)	A≤1,B=0	F1,T2	属第一个判定的取假分支
(4)	A≤1,B≠0	F1,F2	属第一个判定的取假分支
(5)	A=5,X>1	T3,T4	属第二个判定的取真分支
(6)	A=5,X≤1	T3,F4	属第二个判定的取真分支
(7)	A≠5,X>1	F3,T4	属第二个判定的取真分支
(8)	A≠5,X≤1	F3,F4	属第二个判定的取假分支

对于每个判定,要求所有可能的条件取值的组合都必须取到。需要注意的是,这里没有要求第一个判断的 4 个组合再与第二个判断的 4 个组合进行组合。

测试用例可以设计为如下表所示。

测试用例	通过路径	条件取值	覆盖组合号
A=5,B=0,X=4	acef	T1,T2,T3,T4	(1),(5)
A=5,B=1,X=1	abef	T1,F2,T3,F4	(2),(6)
A=1,B=0,X=3	abef	F1,T2,F3,T4	(3),(7)
A=1,B=1,X=1	abdf	F1,F2,F3,F4	(4),(8)

这组测试用例覆盖了所有条件的可能取值的组合,覆盖了所有判定的可取分支,但没有覆盖路径 acdf,不能保证所有的路径被执行,所以条件组合覆盖也有不足之处。

2. 基本路径测试法

在实践中,一个不太复杂的程序,其路径组合可能是一个庞大的数字,要在测试中覆盖所有的路径是不现实的。所以,需要把覆盖的路径数压缩到一定限度内。基本路径覆盖就是这样一种方法,它在程序控制流图的基础上,通过分析控制构造的环路复杂性,导出基本可执行路径的集合,从而设计测试用例的方法。设计出的测试用例要保证在测试中程序的每一条可执行语句至少被执行一次。

1) 程序的控制流图

控制流程图是描述程序控制流的一种图示方式,图 10-4 为基本的控制结构对应的图形符号,其中圆圈称为控制流图的一个结点,它表示一个或多个无分支的语句或源程序;箭头称为边或路径,代表控制流。

如图 10-5 所示,是一个程序的流程图与其对应的控制流图。这里假定图中用菱形框表示的判定条件内没有复合条件。类似于流程图中的流线,一条边必须终止于一个结点。在选择或者多分支结构中,即使汇聚处没有执行语句也应该添加一个汇聚结点。由边和结点圈定的范围称为区域,当对区域计数时,图形外的区域也应记为一个区域。

如果判断中的条件表达式是复合条件,即条件表达式是由一个或多个逻辑运算符(OR、AND、NOR)连接的复合条件表达式,则需要改为一系列只有单个条件的嵌套的判断。如图 10-6 所示,条件语句 A<0 AND B>0 中,条件 A<0 和 B>0 各有一个只有单个条件的判断结点。

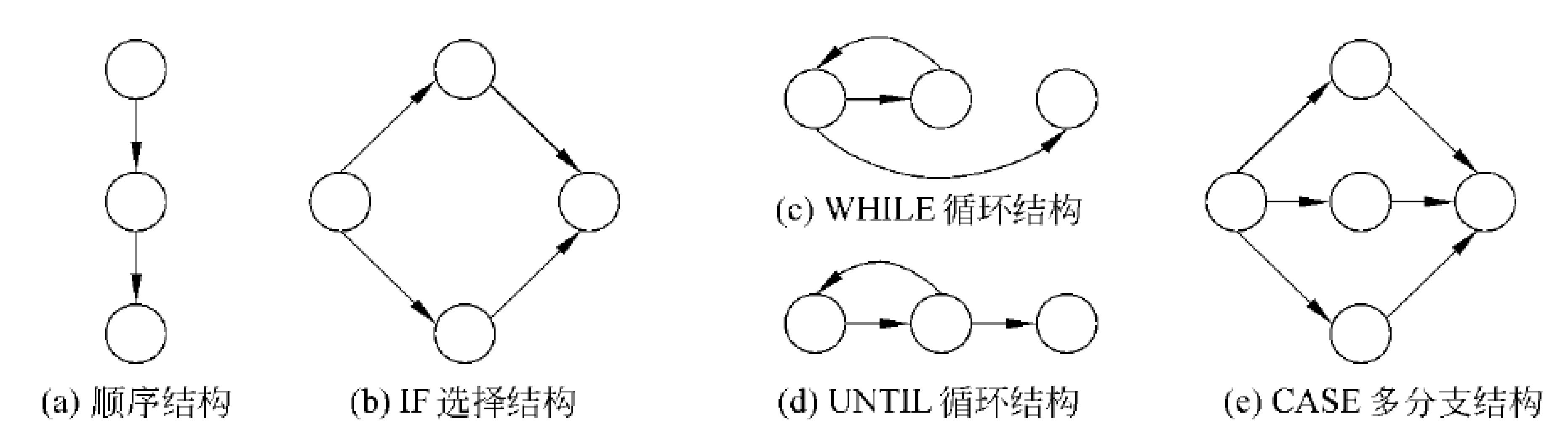


图 10-4　控制流图的图形符号

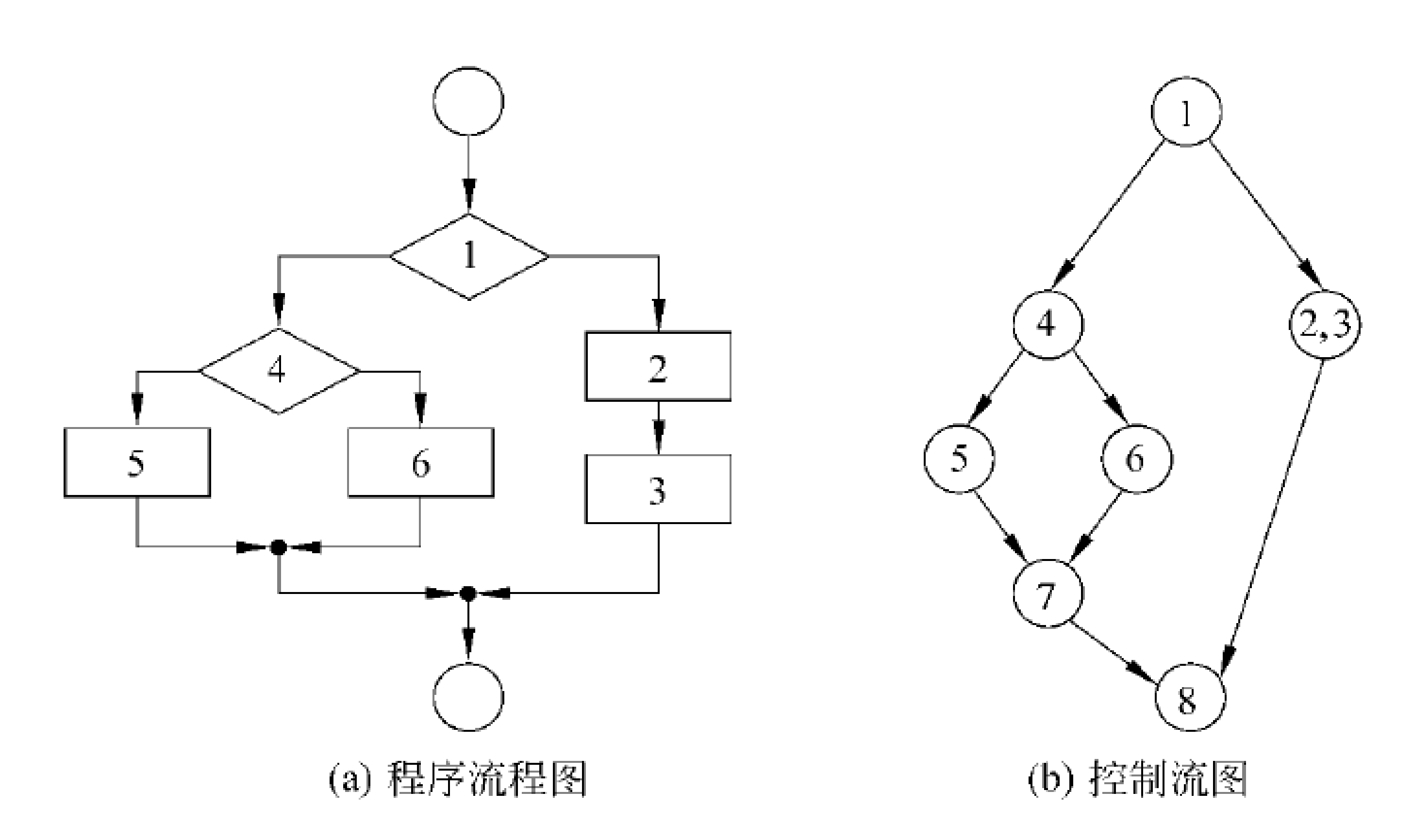


图 10-5　程序流程图和对应的控制流图

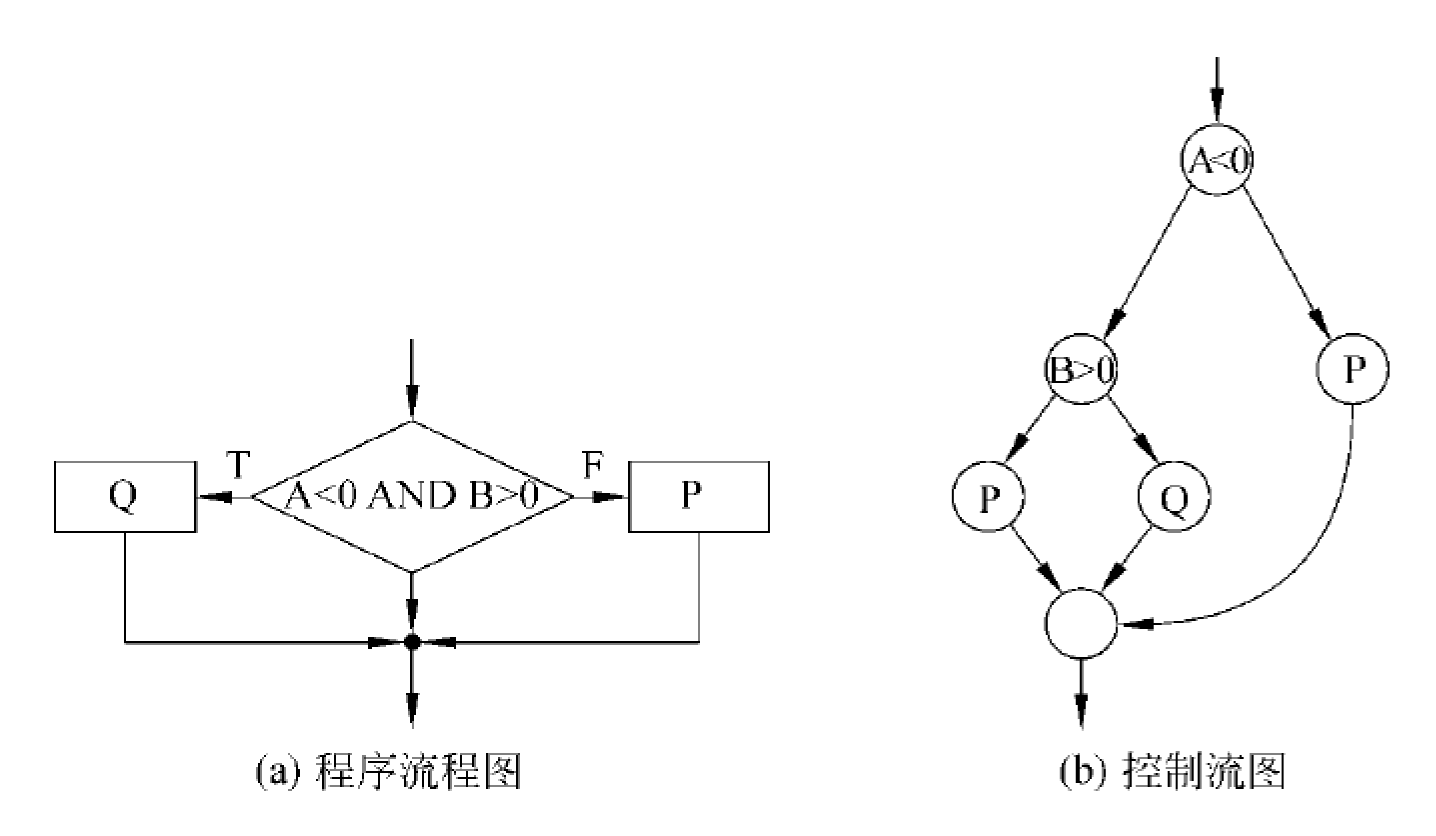


图 10-6　复合逻辑下的控制流图

2）程序环路复杂性

程序的环路复杂性是一种为程序逻辑复杂性提供定量测度的软件度量，通过对程序控制流图的分析和判断来计算模块的复杂性，从程序的环路复杂性可导出程序基本路径集合中的独立路径条数，这是确保程序中每个可执行语句至少执行一次所必需的测试用例数目的上界。

环路复杂性的计算方法有三种：

① 程序的环形复杂度计算公式为

$$V(G) = m - n + 2$$

其中，m 是程序流图 G 中边的数量；n 是结点的数量。

② 如果 P 是流图中判定结点的个数

$$V(G) = P + 1$$

源代码 IF 语句及 While、For 或 Repeat 循环语句的判定结点数为 1，而 Case 型等多分支语句的判定结点数等于可能的分支数减去 1。

③ 环路复杂度等于流图的区域数。

独立路径是指包括一组以前没有处理的语句或条件的一条路径。从控制流图上看，一条独立路径是至少包含有一条在其他独立路径中从未有过的边的路径。例如在图 10-5 中所示的控制流图，V(G)＝8－7＋2＝3，一组独立的路径如下。

- 路径 1：1—2—3—8；
- 路径 2：1—4—5—7—8；
- 路径 3：1—4—6—7—8。

路径 1、2、3 组成了一个基本路径集，只要设计出的测试用例能够确保这些基本路径的执行，就可以使程序中的每个可执行语句至少执行一次，每个条件的取真和取假分支也能得到测试。环路复杂性是构成基本路径集的独立路径数的上界，即独立路径的条数等于 V(G)，据此也得到了应该设计的测试用例的数目。基本路径集不是唯一的，对于给定的控制流图，可以得到不同的基本路径集。

3）基本路径测试法的步骤

基本路径测试法设计测试用例的步骤如下：

① 以详细设计或源代码作为基础，导出程序的控制流图。

② 计算得到控制流图 G 的环路复杂性 V(G)。

③ 确定线性独立路径的基本集合。

④ 生成测试用例，确保基本路径集中的每条路径执行。

下面以图 10-1 所举示例为例，说明测试用例的设计过程。

第一步，画出程序控制流图，如图 10-7 所示。

第二步，计算环路复杂性。

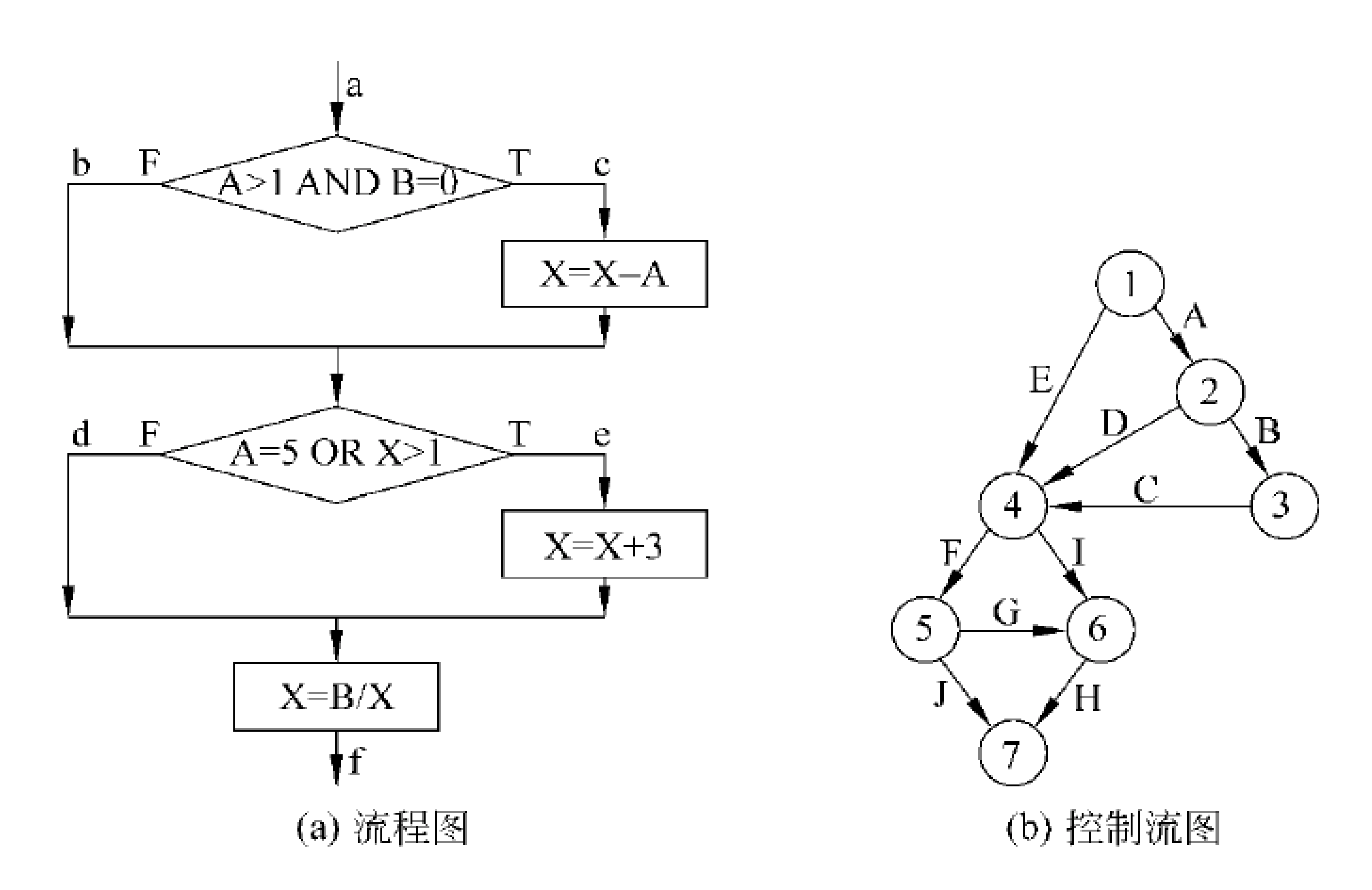
根据流图可以算出：

$$V(G) = m - n + 2 = 10 - 7 + 2 = 5$$

第三步，确定独立路径集合。

所计算出的环路复杂性就是该图已有的独立路径数，5 条路径如下。

- 路径 1：①—②—③—④—⑤—⑦(A—B—C—F—J)；
- 路径 2：①—②—③—④—⑤—⑥—⑦(A—B—C—F—G—H)；
- 路径 3：①—②—③—④—⑥—⑦(A—B—C—I—H)；
- 路径 4：①—②—④—⑤—⑦(A—D—F—J)；
- 路径 5：①—④—⑤—⑦(E—F—J)。



(a) 流程图　　(b) 控制流图

图 10-7　被测程序的流程图和控制流图

那么，其他的路径如：①—④—⑤—⑥—⑦(E—F—G—H)是独立路径吗？答案是否定的，因为上面的 5 条路径已经包括了所有的边，这条路径已经不包含没有用过的边了。

第四步，生成测试用例，确保基本路径集中的每一条路径的执行。

- 路径 1：A=2，B=0，X=1；
- 路径 2：A=2，B=0，X=2；
- 路径 3：A=5，B=0，X=1 或 2；
- 路径 4：A=2，B=1，X=1；
- 路径 5：A=1，B=0 或 1，X=1。

在第三步中，独立路径数也可以为 3。

- 路径 1：①—②—③—④—⑤—⑦(A—B—C—F—J)；
- 路径 2：①—②—④—⑥—⑦(A—D—I—H)；
- 路径 3：①—④—⑤—⑥—⑦(E—F—G—H)。

这是因为，环路复杂性的计算结果表示我们只要最多 5 个独立路径就可以达到基本路径覆盖，它表示的是所取独立路径数的上界，所以独立路径数不是一定要取 5。但需要注意的是，独立路径数越简化代表测试越少，这样程序的安全性就越低了。

3. 数据流测试

数据流测试是用控制流程图对变量的定义和引用进行分析，查找出未定义的变量或定义了而未使用的变量，这些变量可能是拼错的变量、变量混淆或丢失了语句。数据流测试一般使用工具进行。

数据流测试通过一定的覆盖准则，检查程序中每个数据对象的每次定义、使用和消除的情况。

数据流测试步骤：

① 将程序流程图转换成控制流图。

② 在每个链路上标注对有关变量的数据操作的操作符号或符号序列。

③ 选定数据流测试策略。

④ 根据测试策略得到测试路径。

⑤ 根据路径可以获得测试输入数据和测试用例。

动态数据流异常检查在程序运行时执行，获得的是对数据对象的真实操作序列，克服了静态分析检查的局限，但动态方式检查是沿着与测试输入有关的一部分路径进行的，检查的全面性和程序结构覆盖有关。

4. 程序插桩

程序插桩是向使被测程序在保持原有逻辑完整性的基础上，插入所谓“探针”的概念，以便获取程序的控制流和数据流信息，并可估计软件测试的故障覆盖率。

如果想了解一个程序在执行期间所有可执行语句的覆盖情况，或是每个语句的实际执行次数，最好的办法是利用插桩技术。

设计程序插桩时需要考虑的基本问题有以下三个方面：

- 需要探测哪些信息；
- 在程序中什么部位设置探测点；
- 需要设置多少个探测点。

程序插桩由于数据记录量大，手工进行程序插桩测试将是一件很烦琐的事，因此这种测试多借助于自动化工具进行。

5. 域测试

域测试(domain testing)是一种基于程序结构的测试方法。现已经发展成为一个模块测试的有效方法。

域测试将程序错误分为域错误、计算型错误和丢失路径错误三种。如果程序的控制流有错误，对于某一特定的输入可能执行的是一条错误路径，这种错误称为域错误；计算型错误是指如果对于特定输入执行的是正确路径，但由于赋值语句的错误致使输出结果不正确；丢失路径错误是指由于程序中某处少了一个判定谓词而引起的错误。

域测试是主要针对域错误进行测试。“域”是指程序的输入空间，域测试正是在分析输入空间的基础上，完成域的分类、定义和验证，从而对各种不同的域选择适当的测试点(用例)进行测试。

域测试是要判别程序对输入空间的划分是否正确。该方法限制太多，如程序中不能出现数组、子函数、分支谓词是简单谓词等，并且还涉及到多维空间的概念，使用不方便，供有特殊要求的测试使用。

6. 程序变异

程序变异测试是一种错误驱动测试。该方法是针对某类特定程序错误进行的，即专门测试某类错误是否存在。

程序变异测试的基本原理是，在程序的语句中作某些变更，例如将关系运算符“＞”用

“<”替换，使之成为一个新的程序，每个新程序称为原来程序的变异体，从而模拟软件中的各种缺陷，然后根据已有的测试数据，运行变异体，比较变异体和原程序的运行结果：如果两者不同，就称该测试数据将该变异体杀死了。导致变异体不能被杀死的原因有两个：

（1）测试数据集还不够充分，通过扩充测试数据集便能将该变异体杀死。

（2）该变异体在功能上等价于原程序，称这类变异体为等价变异体。杀死变异体的过程一直执行到杀死所有变异体或变异充分度已经达到预期的要求。变异充分度是已杀死的变异体数目与所有已产生的非等价变异体数目的比值。本方法主要应用于测试工具。

7. 符号测试

符号测试的基本思想是允许程序输入的不仅仅是具体的数值数据，而且包括符号值，符号值可以是基本符号变量值，也可以是这些符号变量值的一个表达式。这样在执行程序过程中以符号的计算代替了普通测试执行中对测试用例的数值计算，所得到的结果自然是符号公式或符号谓词，即普通测试执行的是算术运算，符号测试执行的是代数运算。

符号测试可以看做是程序测试与程序验证的一个折中方法，一方面，它沿用了传统的程序测试方法，通过运行被测程序来检验它的可靠性；另一方面，由于一次符号测试的结果代表了一大类普通测试的运行结果，实际上是证明了程序接受此类输入，所得输出是正确的还是错误的。

使用符号测试方法，关键在于开发出比传统的编译器功能更强，能够处理符号运算的编译器或解释器。

10.2.3 黑盒测试用例设计方法

1. 等价类划分法

等价类划分是功能测试用例设计中一种重要的、常用的设计方法，使用这一方法时，完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例。

等价类划分法是把所有可能的输入数据，即程序的输入数据集合划分为若干个子集，然后从每个子集中选取少数代表性数据当作测试用例。每一子集的代表性数据在测试中的作用等价于这一子集中的其他值。使用等价类划分设计测试用例要经过划分等价类和确定测试用例两步。

1）划分等价类

等价类是指某个输入域的子集合，在该子集合中，各个输入数据对于揭露程序中的错误都是等效的。测试某等价类的代表值就等价于对这一类其他值的测试。如果某个等价类中的一个输入条件作为测试数据进行测试查出了错误，那么使用这一等价类中的其他输入条件进行测试也会查出同样的错误，反之亦然。

因此，可以把全部输入数据合理的划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。这些等价类可分为两种：有效等价类和无效等价类。有效等价类是指对于程序的规格说明来说，

是合理的，有意义的输入数据集合。使用有效等价类可以检验程序是否满足需求规格说明所规定的功能和性能；无效等价类和有效等价类相反，是指对于程序的规格说明来说，是不合理的，无意义的输入数据集合。使用无效等价类，可以鉴别程序异常情况的处理。

在设计测试用例时，要同时考虑这两种等价类，因为软件不仅要接收合理的数据，也要能经受意外的考验，这样的测试才能保证软件的可靠性。

如何确定等价类是等价类划分法的关键，需要对规格说明书进行详细分析，找出各个输入条件。下面给出划分等价类的几个原则：

① 如果输入条件规定了取值范围，或值的个数，则可以确立一个有效等价类和两个无效等价类。如：程序输入条件为 1～99 的整数 x，则有效等价类为 $1 \leqslant x \leqslant 99$；两个无效等价类为 $x<1$ 和 $x>99$。

② 如果输入条件规定了输入值的集合，或者是规定了“必须如何”的条件，这时可确立一个有效等价类和一个无效等价类。如：输入条件为 $x=a$，则有效等价类为 $x=a$；无效等类为 $x \neq a$。

③ 如果输入条件是一个布尔量，则可以确定一个有效等价类和一个无效等价类。如：程序输入条件为 BOOL x=true，则有效等价类为 x=true，无效等价类为 x=false。

④ 如果规定了输入数据的一组值（假定 n 个），而且程序要对每个输入值分别处理，可确立 n 个有效等价类和一个无效等价类。如：程序输入条件为 x，取值于一个固定的枚举类型{1,2,3}，程序对这 3 个数值分别进行处理，对于其他的数值采用默认处理方式，则有效等价类为 $x=1$、$x=2$、$x=3$，无效等价类为 $x \neq 1$、2、3 的值的集合。

⑤ 如果规定了输入数据必须遵守的规定，则可以确立一个有效等价类（符合规则）和若干无效等价类（从不同角度违反规则）。如：输入条件规定“必须输入简体中文”，有效等价类就是满足条件的输入的集合，若干个无效等价类：字母、数字、标点、特殊字符等。

⑥ 如果确知，已划分的等价类中各个元素在程序中的处理方式不同，则应将此等价类进一步划分成更小的等价类。

在确立了等价类之后，建立等价类表，列出所有划分出的等价类，如表 10-2 所示。

表 10-2　等价类表

输入条件	有效等价类	无效等价类
⋮	⋮	⋮

2）确立测试用例

根据已列出的等价类表，按以下三步确立测试用例：

① 为每一个等价类规定一个唯一的编号。

② 设计一个测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类。重复这一步，直到所有的有效等价类都被覆盖为止。

③ 设计一个测试用例，使其仅覆盖一个尚未被覆盖的无效等价类，重复这一步，直到所有的无效等价类都被覆盖为止。

例 10-1　某企业招工，要求报名者的出生日期在 1970 年 1 月～1990 年 12 月之间，

企业的人事管理系统需要输入报名者的出生日期，规定日期由 6 位数字字符组成，前 4 位表示年，后 2 位表示月。出生年月不在规定范围内的，系统将拒绝接受，并显示“年龄不合格”的出错信息。试用等价类划分法设计测试用例，来测试系统的“年龄检查功能”。

第一步：划分等价类。

可以划分为 3 个有效等价类，7 个无效等价类，如表 10-3 所示。

表 10-3　实例的等价类表

输 入 条 件	有效等价类		无效等价类	
日期的类型及长度	6 位数字字符	①	有非数字字符 少于 6 个数字字符 多于 6 个数字字符	② ③ ④
年份范围	在 1970～1990 之间	⑤	小于 1970 大于 1990	⑥ ⑦
月份范围	在 01～12 之间	⑧	等于 0 大于 12	⑨ ⑩

第二步：确立测试用例。

首先为每一个等价类规定一个唯一的编号。

为了减小用例数目，设计测试用例时，应该尽可能多的覆盖有效等价类，但是一个测试用例只能对应一个无效等价类。例如在表 10-3 中列出了 3 个有效等价类，编号为①、⑤、⑧，可公用一个测试用例，例如：

测试数据	期望结果	覆盖的有效等价类
198002	输入有效	①、⑤、⑧

为每一个无效等价类设计一个测试用例，如下：

测试数据	期望结果	覆盖的无效等价类
1980ab	输入无效	②
1980	输入无效	③
19800229	输入无效	④
196502	年龄不合格	⑥
199202	年龄不合格	⑦
198000	输入无效	⑨
198014	输入无效	⑩

让几个有效等价类公用一个测试用例，可以减少测试次数，有利而无弊；但若几个无效等价类合用一个测试用例，就可能使错误漏检。就上例而言，假定把 196902（对应无效等价类⑥）和 198000（对应无效等价类⑨）合并为一个测试用例，例如 196900，即使在测试过程中程序显示出“年龄不合格”的信息，仍不能证明程序对月份为 00 的输入数据也具有识别和拒绝接受的功能。再进一步讲，其实在第一步“划分等价类”时，就应防止有意或无

意地将几个独立的无效等价类写成一个无效等价类。例如，若在上例中把⑨、⑩两个无效等价类合并写成“等于 0 或大于 12”，则与之相应的测试用例也将从原来的 2 个减为 1 个，对程序的测试就不够完全了。

2. 边界值分析法

长期的测试工作经验告诉我们，大量的错误是发生在输入或输出范围的边界上，而不是发生在输入输出范围的内部。因此，针对各种边界情况设计测试用例，能取得良好的测试效果，可以查出更多的错误。

在上一例题中，为了只接受年龄合格的报名者，程序中可能设有语句：

```
if(197001<=value(birthdate)<=199012)
then
    read (birthdate)
else
    write "invalid age"
```

但如果编码时把以上语句中的“<=”误写为“<”，用前一例中的测试用例就不能发现这种错误。所谓边界值分析，就是要把测试的重点放在边界上，选取正好等于、刚刚大于和刚刚小于边界值的数据为测试数据，并据此设计出相应的测试用例。

边界值分析法不仅重视输入条件边界，而且也适用于输出域测试用例。在设计测试用例时，应遵循以下几条原则：

① 如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。如以 a 和 b 为边界，测试用例应当包含 a 和 b，以及略大于 a 和略小于 b 的值。

② 如果输入条件规定了值的个数，则用最大个数，最小个数，比最大个数多 1，比最小个数少 1 的数作为测试数据。如一个老师在指导毕业设计时，必须指导 1～5 个学生，则可选人数为 0 个、1 个、5 个、6 个作为测试数据。

③ 根据规格说明的每一个输出条件，使用前面的原则 1、原则 2。例如被测程序是一个求和的函数 SUM(X，Y)，规定其输出范围是 10～20，则在选择测试用例时，X，Y 的取值应使输出值达到边界值及其左右的值，如 X、Y 选取 (5，4)、(6，4)、(10，10)、(10，11)作为测试数据。

④ 如果程序的规格说明给出的输入域或输出域是有序集合(如有序表，顺序文件等)，则应选取集合的第一个元素和最后一个元素作为测试用例。

⑤ 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。

⑥ 分析规格说明，找出其他可能的边界条件。

边界值分析法与等价类划分的区别为，等价类划分法在每一个等价类中任选一个数据作为代表进行测试，而边界值分析要把等价类的每个边界作为测试数据。所以说，边界值分析法是对等价类划分法的补充，在测试中，常将两种方法结合起来共同使用，产生一套完整的测试用例集。

例 10-2 某学生成绩管理系统中,增加学生信息时要求录入学生的各科分数(不计小数点),学生分数的录入范围是 0≤N≤100,试针对录入的分数,设计一套较为完整的测试数据集。

有效等价类有一个:0≤N≤100;无效等价类有两个:N<0,N>100;边界值有两个:0、100。

有效数据的等价输入值选一个,如 75;无效数据的等价输入值选两个,如-999 和 999。边界值附近的值选四个,即-1、1、99 和 101。所以一个较为完整的测试数据集合是{-999,-1,0,1,75,99,100,101,999}。

3. 判定表驱动法

判定表(Decision Table)是分析和表达较为复杂逻辑条件下软件状态和行为的有效工具。利用判定表能够设计出完整的测试用例集合,能够将复杂的问题按照各种可能的情况罗列出来,使测试内容变得简单明了并避免了遗漏。

判定表通常由 4 个部分组成,如图 10-8 所示。



图 10-8 判定表组成

- 条件桩(Condition Stub):列出问题的所有条件,不强调所列条件的次序关系。
- 动作桩(Action Stub):列出问题规定的可以采取的操作。
- 条件项(Condition Entry):列出针对条件桩的取值,即在所有可能情况下的真假值。
- 动作项(Action Entry):列出在条件项的各种取值情况下应采取的动作。

任何一个条件组合的特定取值及其相应要执行的操作,可视为一条规则。表现在判定表中,贯穿条件项和动作项的一列就是一条规则。显然,一条规则就是一个测试用例,判定表列出多少组条件取值,也就有多少条规则和相应数量的测试用例。

判定表的建立依据于软件的规格说明,一般按照以下 5 个步骤进行分析设计:

① 确定规则的个数。若条件数为 n,规则个数应为 2 的 n 次方。

② 列出所有的条件桩和动作桩。

③ 填入条件项。

④ 填入动作项,得到初始判定表。

⑤ 简化判定表,合并相似规则。

- 若表中有两条以上规则具有相同的动作,并且在条件项之间存在极为相似的关系,便可以合并。
- 合并后的条件项用符号“-”表示,说明执行的动作与该条件的取值无关,称为无关条件。

下面给出适合于使用判定表设计测试用例的条件:

① 规格说明以判定表形式给出,或是很容易转换成判定表。

② 条件的排列顺序不会影响执行哪些操作。

③ 规则的排列顺序不会影响执行哪些操作。

④ 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。

⑤ 如果某一规则的条件得到满足，将执行多个操作，这些操作的执行与顺序无关。

例 10-3 某学生成绩管理系统，要求“对平均成绩在 90 分以上，且没有不及格科目的学生，或班级成绩排名在前 5 位的学生，在程序中将学生的姓名用红色标识”，请建立该场景的判定表。

① 确定规则的个数。这里有 3 个条件，每个条件有两个取值，故应有 $2^3=8$ 种规则。

② 列出所有的条件项和动作项，见表 10-4。

表 10-4 所有条件项和动作项

条件	平均成绩是否大于 90
	是否没有不及格的科目
	成绩排名是否在前 5 名
动作	姓名用红色标识
	其他处理

③ 填入条件项、动作项，得到初始判定表(见表 10-5)。

表 10-5 初始判定表

		1	2	3	4	5	6	7	8
条件	平均成绩大于 90	Y	Y	Y	Y	N	N	N	N
	没有不及格的科日	Y	Y	N	N	Y	Y	N	N
	成绩排名在前 5 名	Y	N	Y	N	Y	N	Y	N
动作	姓名用红色标识	X	X	X		X		X	
	其他处理				X		X		X

④ 合并相似规则，得到优化后判定表(见表 10-6)。

表 10-6 判定表

		1	2	3	4	5
条件	平均成绩大于 90	Y	Y	Y	N	N
	没有不及格的科目	Y	N	N	—	—
	成绩排名在前 5 名	—	Y	N	Y	N
动作	姓名用红色标识	X	X		X	
	其他处理			X		X

4. 因果图

等价类划分方法和边界值分析法，主要是针对单个输入数据来设计测试用例的，没有考虑到输入情况的各种组合，也没有考虑到各个输入情况之间的相互制约关系。这样虽

然各种输入条件可能出错的情况已经测试到了，但多个输入条件组合起来可能出错的情况却被忽视了。如果在测试时必须考虑输入条件的各种组合，则可能的组合数将是天文数字，因此必须考虑采用一种适合于描述多种条件的组合、相应的产生多个动作的形式来进行测试用例的设计，这就需要利用因果图。

因果图法是从自然语言书写的程序规格说明的描述中找出因（输入条件）和果（输出或程序状态的改变），通过因果图转换为判定表，然后为判定表的每一列设计一个测试用例。采用因果图方法能够高效地选择测试用例，同时还能指出程序规格说明中存在的不完整性和二义性。

利用因果图生成测试用例，一般需要经过以下几个步骤：

① 分析程序规格说明，引出原因（输入条件）和结果（输出结果），并给每个原因和结果赋予一个标识符。原因常常是输入条件或是输入条件的等价类，而结果是输出条件。

② 分析程序规格说明中语义的内容，并将其表示成连接各个原因和各个结果的"因果图"。

③ 在因果图上标明约束条件。

④ 通过跟踪因果图中的状态条件，把因果图转换成有限项的判定表。

⑤ 把判定表中每一列表示的情况生成测试用例。

如图 10-9 所示，通常在因果图中用 C_i 表示原因，用 E_i 表示结果。各结点表示状态，可取值 0 或 1。0 表示某状态不出现，1 表示某状态出现。主要的原因和结果之间的关系有：

① 恒等。表示原因与结果之间一对一的对应关系。若原因出现，则结果出现，若原因不出现，则结果也不出现。

② 非。表示原因与结果之间的一种否定关系。若原因出现，则结果不出现；若原因不出现，则结果出现。

③ 或。表示若几个原因中有一个出现，则结果出现，只有当这几个原因都不出现时，结果才不出现。

④ 与。表示若几个原因都出现，结果才出现，若几个原因中有一个不出现，结果就不出现。

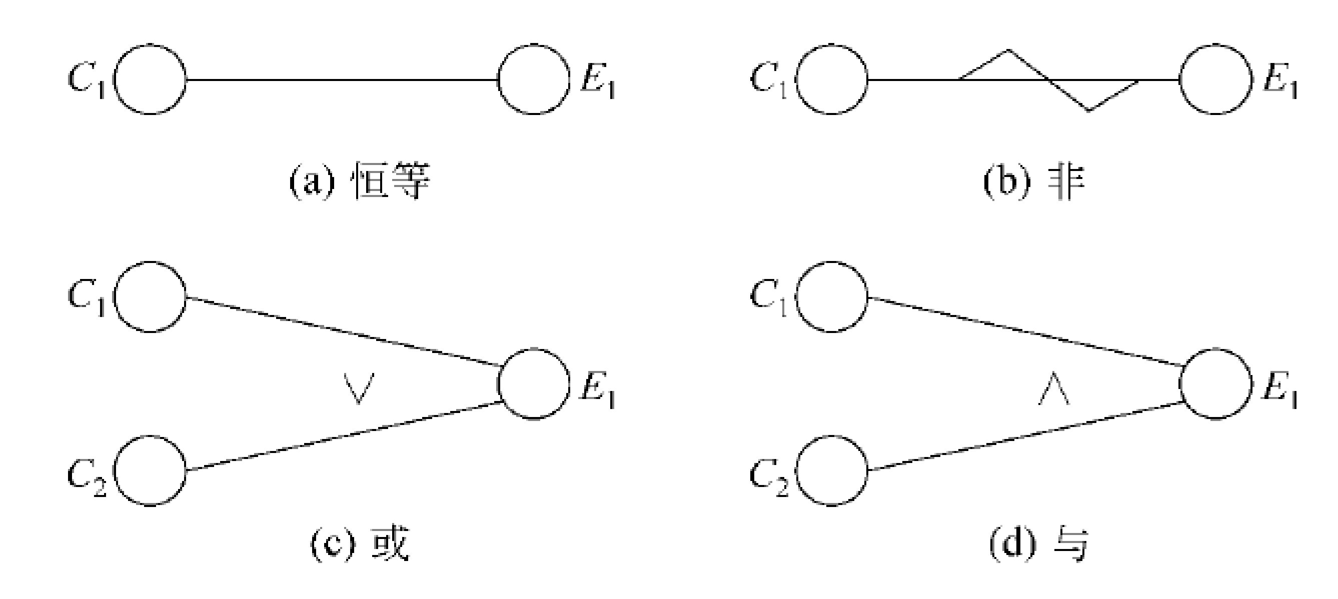


图 10-9　因果图逻辑符号表示方法

为了表示原因与原因之间、结果与结果之间可能存在的约束条件，在因果图中可以附加一些表示约束条件的符号。从输入（原因）考虑，有 4 种约束：互斥、包含、唯一、要求，从输出（结果）考虑，有 1 种约束：屏蔽（如图 10-10 所示）。

(a) E(互斥) (b) I(包含) (c) O(唯一) (d) R(要求) (e) M(屏蔽)

图 10-10 约束条件

① E(互斥)。表示 a,b 两个原因不会同时成立,两个中最多有一个可能成立。

② I(包含)。表示 a,b,c 三个原因中至少有一个必须成立。

③ O(唯一)。表示 a,b 当中必须有一个,且仅有一个成立。

④ R(要求)。表示当 a 出现时,b 必须也出现。不可能 a 出现,b 不出现。

⑤ M(屏蔽)。表示当 a 是 1 时,b 必须是 0,而当 a 为 0 时,b 值不定。

对于因果图法,如果需求规格说明中含有输入条件的组合,宜采用本方法,它能有力的帮助我们确定测试用例。有些软件的因果图可能非常庞大,以至于根据因果图得到的测试用例数目非常大,此时就不宜使用本方法。

例 10-4 BPExpress 是一个操作符后置数学表达式的计算器的简单实现,由用户按照规则输入包含加、减、乘、除这四类运算符的数学表达式,由 BPExpress 对表达式进行分析计算并给出运算结果。

例如:用户输入

```
32 8+3* 4/
```

随着处理过程的进行,堆中数据变化如下:

```
32
32 8
40
40 3
120
120 4
30
```

即经 BPExpress 计算后,最终结果为:

```
30
```

BPExpress 可以对计算过程中的堆的数据变化情况予以监控,并将监控结果输出到指定文件中。设计测试用例检查记录的内容是否正确。BPExpress 监控输出处理流程(如图 10-11 所示)是这样的:进行计算时,首先检查是否进行"堆监控",如果需要堆监控则检查监控文件路径是否设置(如果未设置,则提示用户设置路径),如已设置继续检查路径合法性(如果不合法,则提示重新设置路径),如果合法,则检查监控文件是否存在,如果存在则写入新的监控信息,如果不存在则创建新的监控文件并写入新的监控信息。

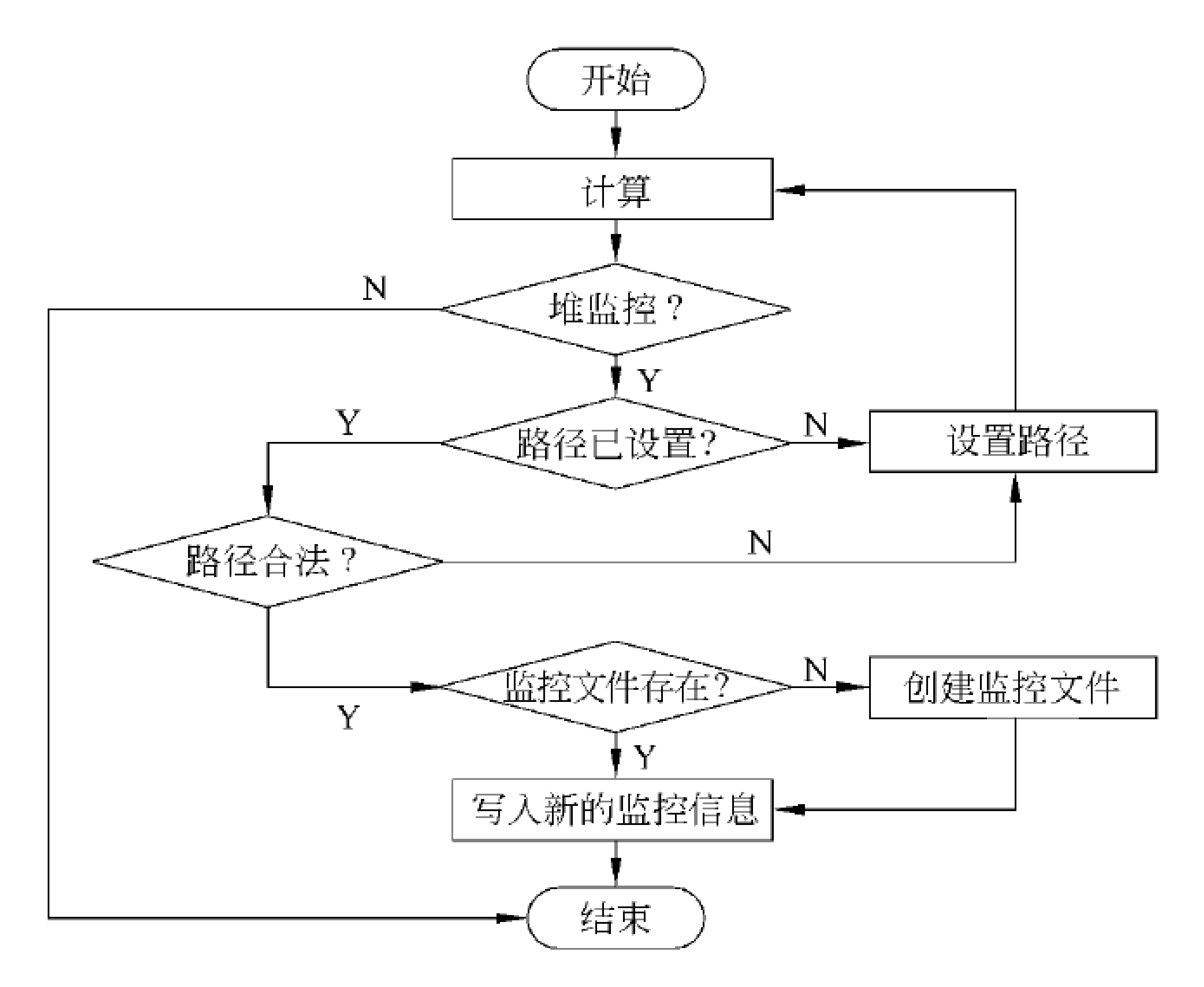


图 10-11　堆监控控制流程图

解答：分析 BPExpress 堆监控的这一说明，可以列出原因和结果，见表 10-7。

表 10-7　原因和结果

序号	原　　因	结　　果
1	启动堆监控	输出监控信息
2	禁止堆监控	不输出监控信息
3	路径存在	创建文件
4	路径不存在	设置路径
5	监控文件不存在	
6	监控文件存在	

根据原因和结果，可以画出一个因果图，如图 10-12 所示。

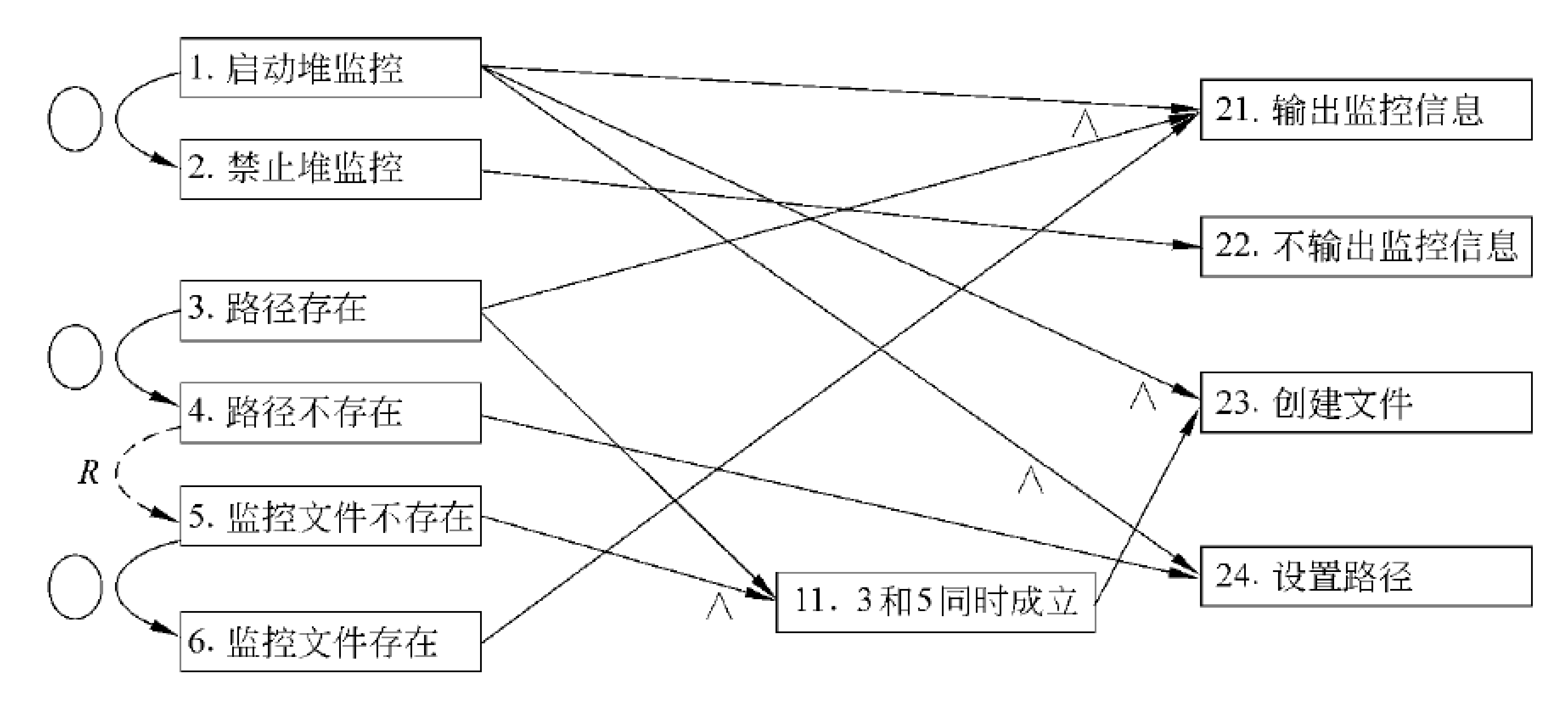


图 10-12　BPExpress 堆监控因果图

将因果图转化为判定表。如表 10-8 所示，每一列可作为确定测试用例的依据。

表 10-8　BPExpress 堆监控判定表

条件	1	1	1	1	0	
	2	0	0	0	1	
	3	0	1	1	0	
	4	1	0	0	0	
	5	1	0	1	0	
	6	0	1	0	0	
中间		0	0	1	0	
结果	21	0	1	0	0	
	22	0	0	0	1	
	23	0	0	1	0	
	24	1	0	0	0	

5. 错误推测法

错误推测法也称为猜错法，是基于经验和直觉推测程序中所有可能存在的各种错误，有针对性的设计测试用例的测试方法。

错误推测法的基本思想是列举出程序中所有可能存在的错误和容易发生错误的特殊情况，从中作出选择，以设计测试用例。例如：

- 若在单元测试中程序模块测试出现了错误，在后期功能测试中可以列出这些可能出现问题的地方，设计相应的测试用例；
- 根据前一个版本中发现的常见错误，有针对性的为当前版本设计测试用例；
- 在应用软件中可能出错的环节，如 C++ 程序的内存泄露、Web 程序的 session 失效问题、JavaScript 字符转义等一些常见的普通问题，选择性的设计测试用例。

错误推测法针对性强，可以直接切入可能的错误，直接定位，是一种非常实用、有效的方法。但是它需要丰富的经验和专业知识。

6. 正交实验

正交实验法是从大量的实验点中挑出适量的、有代表性的点，应用正交表，合理地安排实验的一种科学的实验设计方法。

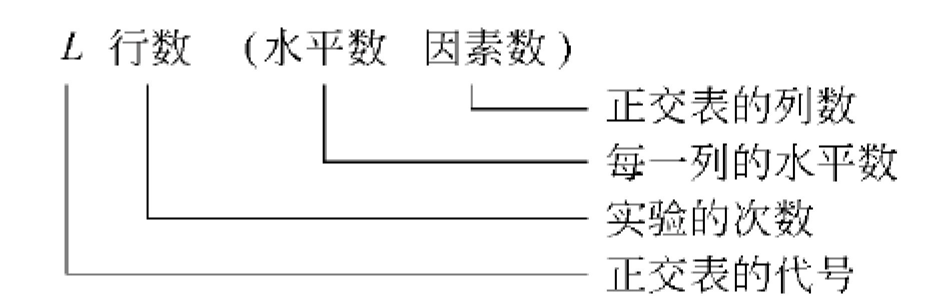
利用正交实验法来设计测试用例时，首先要根据被测软件的规格说明书找出影响功能实现的操作对象和外部因素，把它们当作因子，而把各个因子的取值当作状态，生成二元的因素分析表。然后，利用正交表进行各因子的状态的组合，构造有效的测试输入数据集，这样得出的测试用例的数目将大大减少。

在正交实验法中，有两个需要理解的概念：

(1) 因素。在一项实验中，凡欲考察的变量称为因素(变量)。

（2）水平。在实验范围内，因素被考察的值称为水平（变量的值）。

正交表的表示形式如下：



- L 表示正交表。
- 行数（runs）表示正交表中的行的个数，即需要做的实验的次数。
- 因素数（factors）表示正交表的总列数，表示最多可以安排的因素个数。
- 水平数（levels）表示任何单个因素合法取值个数的最大个数。

例如，$L_4(2^3)$，其中 3 为此表列的数目（最多可安排的因素数），2 为因素的水平数，4 为此表行的数目（实验次数），对应的正交表如下，其中 0 对应因素的第一种取值，1 对应第二种取值。

0	0	0
0	1	1
1	0	1
1	1	0

上表中，各列的水平数相同，我们称为等水平正交表。一个正交表中也可以各列的水平数不相等，我们称它为混合水平正交表，例如 $L_8(2^4 4^1)$，表示有 4 列是 2 水平的，有 1 列是 4 水平的，8 为此表的行的数目（实验次数），对应的正交表如下。

0	0	0	0	0
0	0	1	1	2
0	1	0	1	1
0	1	1	0	3
1	0	0	1	3
1	0	1	0	1
1	1	0	0	2
1	1	1	1	0

正交表具有以下两项性质：

（1）每一列中，不同的数字出现的次数相等。如 $L_4(2^3)$正交表中，任何一列都有 0 与 1，且任何一列中它们出现的次数是相等的，均分别出现 2 次。

（2）任意两列中数字的排列方式齐全而且均衡，任何两列所构成的各有序数对出现的次数相等。如 $L_4(2^3)$正交表中，同一行内的任何两列有序对共有 4 种：（0，0）、（0，1）、（1，0）、（1，1），每种对数出现次数相等，均出现 1 次。

以上两点充分的体现了正交表的两大优越性，即“均匀分散，整齐可比”。通俗地说，

每个因素的每个水平与另一个因素的各个水平各碰一次，这就是正交性。

与通常设计试验的过程类似，正交实验法运用于测试用例优选设计的步骤为：

① 分析所需测试的软件的特点，列出每个独立的测试项。如可将需要测试的软件分解为不同的功能模块及性能项。

② 对某个测试项而言，具体分析影响当前测试项的因素。如测试的是一个函数，则其影响因素可以是函数包含的参数；对于一个 GUI 界面功能测试来说，其影响因素可以是界面中的输入域或者相应的操作。

③ 分析每个影响因素的取值方式。可利用等价类划分法及边界值分析法，确定单个因素的取值方式。

④ 对某个测试项，设计或选择一个适合的正交表，将正交表的列对应于该测试项的影响因素，正交表的水平数对应于影响因素的取值方式，正交表的行数则对应于测试用例的个数，即测试次数；通常我们所选的合适的正交表，应选取行数最少的那个。

⑤ 将正交表中的元素值分别转换为因素的实际取值，逐行选择正交表中的组合数据作为优选后的测试用例，同时加上你认为可疑且没有在表中出现的组合，执行测试。

⑥ 对其他的测试项，重复步骤②～步骤⑤，即可得到整个软件的优选后的测试用例。

由以上分析可得出，在正交实验法中，必须根据被测软件的特点，如影响结果的因素个数及各因素的取值情况，确定与之对应的合适的正交表，然后根据一定的转换规则，由正交表转换成测试用例列表。因此，是否能构造出合适的正交表，关系到正交实验法的成功与否。

正交表的推导过程可以依据 Galois 理论，本书不做介绍，需要时可查阅相关数理统计方面的教材。此处给出部分常用的正交表。

- 二水平正交表：$L_4(2^3)$，$L_8(2^7)$，$L_{12}(2^{11})$，$L_{16}(2^{15})$，…；
- 三水平正交表：$L_9(3^4)$，$L_{27}(3^{13})$，…；
- 四水平正交表，$L_{16}(4^5)$，…；
- 五水平正交表：$L_{25}(5^6)$，…。
- 常用的混合水平正交表有：$L_8(4^1\times2^4)$，$L_{16}(4\times2^{12})$，$L_{16}(4\times2^9)$，$L_{16}(4^4\times2^3)$，$L_{18}(3^6\times6^1)$…

例 10-5 假设有一个系统功能模块有 5 个独立的变量 A、B、C、D、E。变量 A 和 B 都有两个取值（A_1、A_2 和 B_1、B_2）。变量 C 和 D 都有三个可能的取值（C_1、C_2、C_3 和 D_1、D_2、D_3）。变量 E 有六个可能的取值 E_1、E_2、E_3、E_4、E_5、E_6。

① 分析影响因素及因素取值。

影响该功能执行的因素及每个因素的取值方式见表 10-9。

② 选择合适的正交表。

- 表 10-8 中的因素数（变量）≥5；
- 至少有两个因素的水平数（变量的取值）≥2；
- 至少有另外两个因素的水平数≥3；
- 还至少有另外一个因素的水平数≥6；

表 10-9　影响因素及其取值

序号	因素名称	取值方式					
		方式 1	方式 2	方式 3	方式 4	方式 5	方式 6
Factor1	A	A_1	A_2				
Factor2	B	B_1	B_2				
Factor3	C	C_1	C_2	C_3			
Factor4	D	D_1	D_2	D_3			
Factor5	E	E_1	E_2	E_3	E_4	E_5	E_6

根据上述分析，从已经推导出的正交表库中选取最合适的一个，即行数取最少的一个：$L_{18}(3^6\times6^1)$（如表 10-10 所示）。

表 10-10　正交表

	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6	Factor7
Run1	0	0	0	0	0	0	0
Run2	0	0	1	1	2	2	1
Run3	0	1	0	2	2	1	2
Run4	0	1	2	0	1	2	3
Run5	0	2	1	2	1	0	4
Run6	0	2	2	1	0	1	5
Run7	1	0	0	2	1	2	5
Run8	1	0	2	0	2	1	4
Run9	1	1	1	1	1	1	0
Run10	1	1	2	2	0	0	1
Run11	1	2	0	1	2	0	3
Run12	1	2	1	0	0	2	2
Run13	2	0	1	2	0	1	3
Run14	2	0	2	1	1	0	2
Run15	2	1	0	1	0	2	4
Run16	2	1	1	0	2	0	5
Run17	2	2	0	0	1	1	1
Run18	2	2	2	2	2	2	0

③ 变量映射，将正交表中的元素值分别转换为因素的实际取值，形成用例列表（表 10-11）。

- A：$0\rightarrow A_1, 1\rightarrow A_2, 2\rightarrow A_1/A_2$；
- B：$0\rightarrow B_1, 1\rightarrow B_2, 2\rightarrow B_1/B_2$；
- C：$0\rightarrow C_1, 1\rightarrow C_2, 2\rightarrow C_3$；

- D：0→D_1，1→D_2，2→D_3；
- E：0→E_1，1→E_2，2→E_3，3→E_4，4→E_5，5→E_6。

表 10-11　用例表

	Factor1	Factor2	Factor3	Factor4	Factor5	Factor6	Factor7
Run1	A_1	B_1	C_1	D_1	—	—	E_1
Run2	A_1	B_1	C_2	D_2	—	—	E_2
Run3	A_1	B_2	C_1	D_3	—	—	E_3
Run4	A_1	B_2	C_3	D_1	—	—	E_4
Run5	A_1	B_1	C_2	D_3	—	—	E_5
Run6	A_1	B_2	C_3	D_2	—	—	E_6
Run7	A_2	B_1	C_1	D_3	—	—	E_6
Run8	A_2	B_1	C_3	D_1	—	—	E_5
Run9	A_2	B_2	C_2	D_2	—	—	E_1
Run10	A_2	B_2	C_3	D_3	—	—	E_2
Run11	A_2	B_1	C_1	D_2	—	—	E_4
Run12	A_2	B_2	C_2	D_1	—	—	E_3
Run13	A_1	B_1	C_2	D_3	—	—	E_4
Run14	A_2	B_1	C_3	D_2	—	—	E_3
Run15	A_1	B_2	C_1	D_2	—	—	E_5
Run16	A_2	B_2	C_2	D_1	—	—	E_6
Run17	A_1	B_1	C_1	D_1	—	—	E_2
Run18	A_2	B_2	C_3	D_3	—	—	E_1

由上述内容可见，对于该功能模块，只需做 18 次测试就可以充分分析出测试结果了，如果做完全的测试需要 2×2×3×3×6＝216 次试验来完成。借助正交实验法设计软件测试用例，不仅可以有效覆盖整个测试的范围，保证了用例的有效性，而且还大大减少了测试人员的工作量，节约了测试成本。

7. 场景法

现在的软件几乎都是用事件触发来控制流程的，每个事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果形成事件流。这种在软件设计方面的思想也可以引入到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于设计测试用例，同时使测试用例更容易理解和执行。

事件流分为基本流和备选流。基本流描述的是该用例最正常的一种场景，在基本流中系统执行一系列活动步骤来响应参与者提出的服务请求；备选流负责描述用例执行过程中异常的或偶尔发生的一些情况，备选流和基本流的组合应该能够覆盖该用例所有可能发生的场景。场景法就是通过用例场景描述业务操作流程，从用例开始到结束遍历应用流程上所有基本流和备选流。

如图 10-13 所示,图中经过用例的每条路径都用基本流和备选流来表示,直黑线表示基本流,是经过用例的最简单的路径。备选流可用不同的色彩表示,一个备选流可能从基本流开始,在某个特定条件下执行,然后重新加入基本流中(如备选流 1 和 3);也可能起源于另一个备选流(如备选流 2),或者结束用例而不再重新加入到某个流(如备选流 2 和 4)。

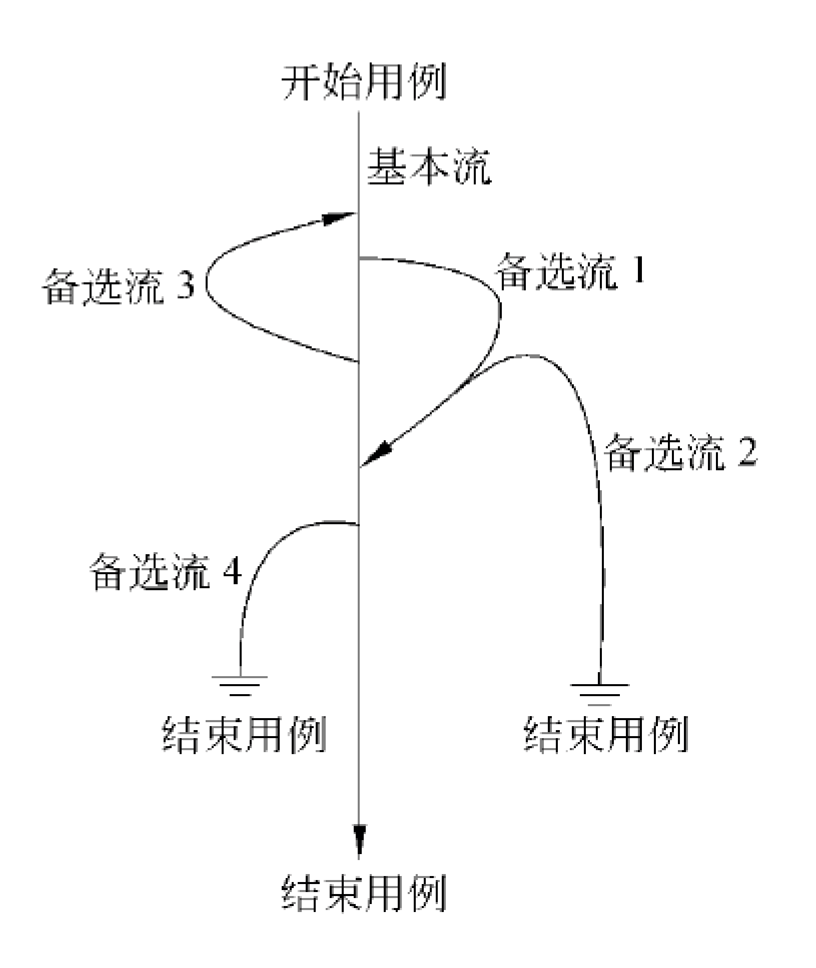


图 10-13 事件流示例

按照上图中每个经过用例的路径,可以确定以下不同的用例场景:

① 场景 1。基本流。

② 场景 2。基本流,备选流 1。

③ 场景 3。基本流,备选流 1,备选流 2。

④ 场景 4。基本流,备选流 3。

⑤ 场景 5。基本流,备选流 3,备选流 1。

⑥ 场景 6。基本流,备选流 3 备选流 1,备选流 2。

⑦ 场景 7。基本流,备选流 4。

⑧ 场景 8。基本流,备选流 3,备选流 4。

注意:为方便起见,场景 5、6 和 8 只考虑了备选流 3 循环执行一次的情况。

下面是场景法的基本设计步骤:

① 根据规格说明,描述出程序的基本流及各项备选流。

② 根据基本流和各项备选流生成不同的场景。

③ 对每一个场景生成相应的测试用例,一般采用矩阵或决策表来确定和管理测试用例。

④ 对生成的所有测试用例重新复审,去掉多余的测试用例,测试用例确定后,对每一个测试用例设定测试数据。

8. 功能图法

一个程序的功能说明通常由动态说明和静态说明组成。动态说明描述了输入数据的次序或转移的次序,静态说明描述了输入条件与输出条件之间的对应关系。对于较复杂的程序,由于存在大量的组合情况,仅仅使用静态说明组成的规格说明对于测试来说往往是不够的,必须用动态说明来补充功能说明。功能图法就是因此而产生的一种测试用例设计方法。

使用功能图法设计测试用例,是借助功能图模型来实现的,用功能图形式化的表示程序的功能说明,并机械地生成功能图的测试用例。功能图模型由状态迁移图和逻辑功能模型组成,分别介绍如下:

状态迁移图用于表示输入数据序列以及相应的输出数据。在状态迁移图中,由输入

数据和当前状态决定输出数据和后续状态。

逻辑功能模型用于表示在状态中输入条件和输出条件之间的对应关系。逻辑功能模型只适合于描述静态说明，输出数据仅由输入数据决定。测试用例则是由测试中经过的一系列状态和在每个状态中必须依靠输入/输出数据满足的一对条件组成。

功能图法其实是一种黑盒、白盒混合的用例设计方法。功能图法中，要用到逻辑覆盖和路径测试的概念和方法，这属于白盒测试方法中的内容。以下提到的逻辑覆盖和路径是功能或系统水平上的，以区别于白盒测试中的程序内部的。

以某“用户登录”功能为例，简单说明状态迁移图的应用。从等待状态开始，会达到成功状态或失败状态，并找出导致这种状态改变的输入条件(如图 10-14 所示)。

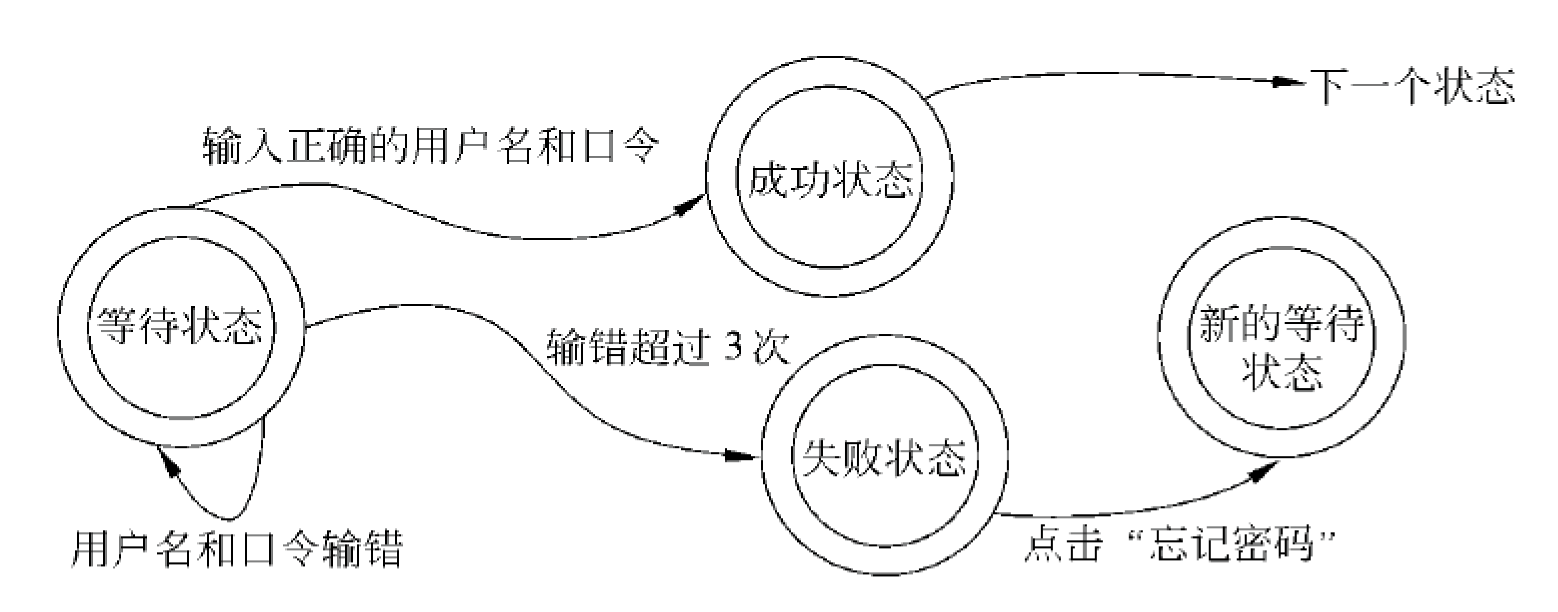


图 10-14　系统登录的状态迁移图

针对同样的例子，也可以用判定表来描述逻辑功能模型，如表 10-12 所示。

表 10-12　判定表

输入	正确的用户名	错误的用户名	错误的用户名	正确的用户名
	错误的密码	正确的密码	错误的密码	正确的密码
输出	0	0	0	1
	错误提示	错误提示	错误提示	
状态	等待重新输入	等待重新输入	等待重新输入	进入新的状态

在状态迁移图中，若用结点代替状态，用弧线代替迁移，状态迁移图就可以转化成一个程序的控制流程图形式。问题就转化为程序的路径测试问题(白盒测试范畴概念)了。在逻辑功能表中，可以根据所有的输入输出以及状态来生成所需要的结点和路径，形成功能图的基本路径组合：

① 局部测试用例由原因组合(输入数据)与对应的结果值(输出数据)构成。

② 整体测试用例，由从初始状态到最后状态的测试路径构成。

9. 测试方法选择的综合策略

测试用例的设计方法不是单独存在的，具体到每个测试项目中都会用到多种方法，每种类型的软件有各自的特点，每种测试用例设计的方法也有各自的特点，针对不同的软件如何利用这些黑盒测试方法是非常重要的，在实际测试中，往往是综合使用各种方法才能

有效地提高测试效率和测试覆盖度。以下是各种测试方法选择的综合策略，可供读者在实际应用过程中参考：

① 首先进行等价类划分，包括输入条件和输出条件的等价类划分，将无限测试变成有限测试，这是减少工作量和提高工作效率最有效的方法。

② 在任何情况下都必须使用边界值分析方法。经验表明，用这种方法设计出的测试用例发现程序错误的能力最强。

③ 用错误推测法追加一些测试用例，这需要依靠测试工程师的智慧和经验。

④ 对照程序逻辑，检查已设计出的测试用例的逻辑覆盖程度，如果没有达到要求的覆盖标准，应当再补充足够的测试用例。

⑤ 如果程序的功能说明中含有输入条件的组合，则一开始就可选用因果图法和判定表驱动法。

⑥ 对于参数配置类的软件，要用正交实验法选择较少的组合方式达到最佳效果。

⑦ 对于业务流清晰的系统，可以利用场景法贯穿整个测试案例过程，在案例中综合使用各种测试方法。

⑧ 功能图法也是很好的测试用例设计方法，可以通过不同时期条件的有效性设计不同的测试数据。

10.3 应用实例讲解

本章前面几节对测试的基本技术做了比较全面的介绍，那么这么多的测试方法在具体的测试案例中如何选择使用呢？事实上，在软件测试领域，不仅仅是测试方法繁多，而且测试类型也是非常多的。为了帮助大家更好的理解和认识，第二篇软件测试分类一节已经整理分析了大多数类型的测试，并从下面四个维度对它们进行了分类（如图 10-15 所示）。

（1）按照测试阶段分类，软件测试可以分为单元测试、集成（或组装）测试、系统测试、验收测试及 Alpha/Beta 测试。

按照测试方式和方法分类，软件测试分为静态测试和动态测试，从大的角度可以分为白盒测试和黑盒测试。

（2）按照测试执行者分类，软件测试有开发测试（第一方测试）、用户测试（第二方测试）和第三方测试。

（3）从测试的关注点不同分类，软件测试有功能测试、接口测试、性能测试、负载测试、强度测试、容量测试、用户界面（UI）测试、安全性测试、配置测试和安装测试等。

这些不同分类的测试之间并不是孤立的关系，它们之间是有联系的，比如单元测试、集成测试从大的分类角度看属于白盒测试的范畴，而系统测试、验收测试主要以黑盒测试为主。另外，我们还知道软件测试是服务于软件质量的，因此从质量角度来看，这些不同种类的测试本质上是服务于不同的质量属性（参见本篇 5.2.2 节有关内容）。

本书将重点讨论的是按照阶段划分的四类测试，即单元测试、集成测试、系统测试和验收测试，因为这四类测试是软件测试的中轴线，也是软件测试的基本功，熟练掌握了这

四类测试，对于其他各类测试的理解能够达到事半功倍的效果。

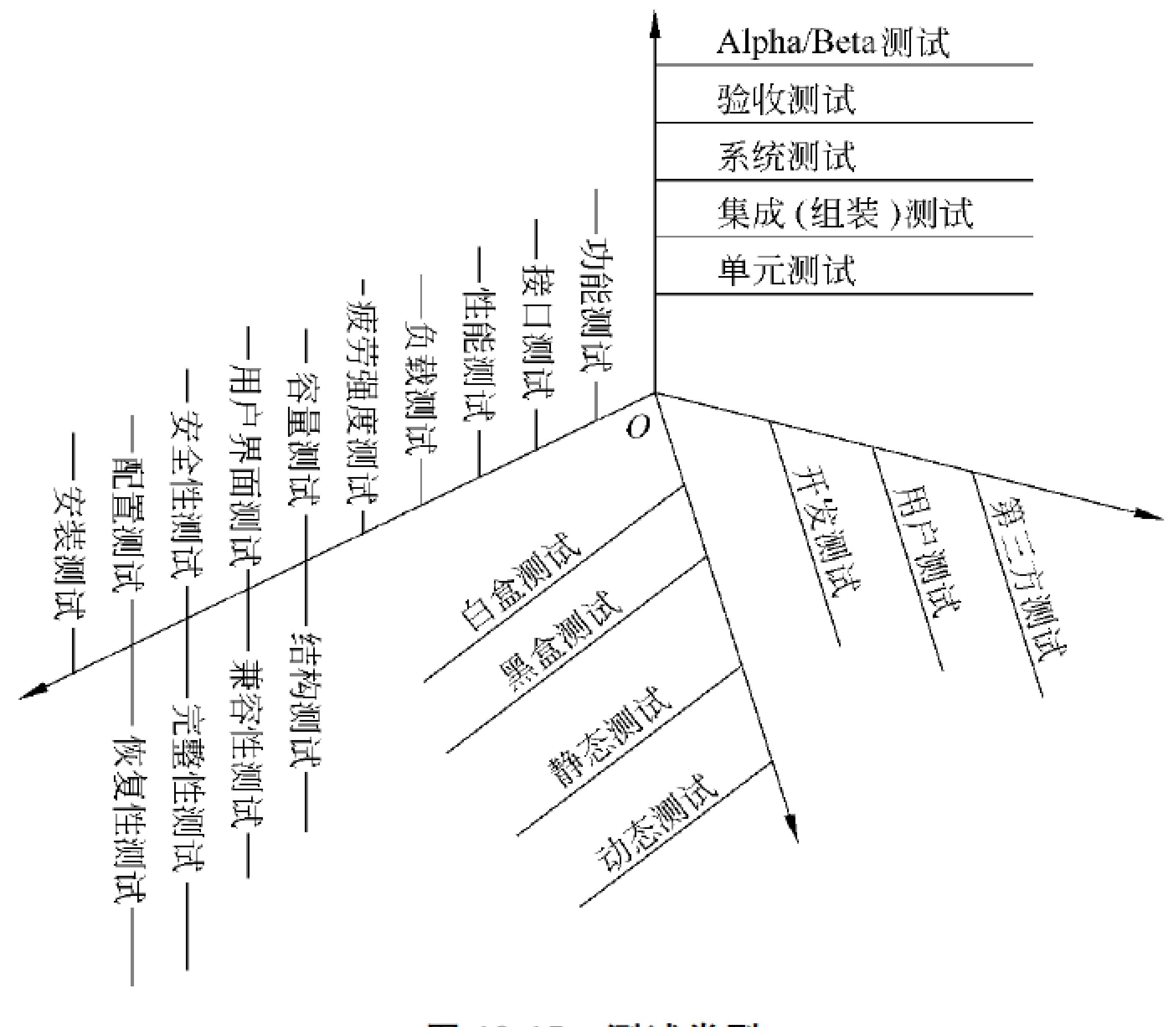


图 10-15　测试类型

10.3.1　单元测试

1. 概述

按照通常的认知顺序，在介绍单元测试之前，仍然是先给出它的定义：单元测试是在软件开发过程中要进行的最低级别的测试活动，在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

单元测试是软件测试中的焦点，这一方面是因为单元测试的难度相比于其他类型的测试要大；另一方面则是针对单元测试存在较大的有关投资收益的争议。但是不管怎么说，单元测试是测试领域离代码最近的测试类型，在开发群体中比较容易获得认可。近年来，随着开发技术的快速发展和对软件质量的呼声日高，越来越多的人或机构开始关注单元测试，单元测试技术也因此获得了极大提高，开始展露其实用性特征。

要讲单元测试，首先应该明确一个问题，那就是一个单元应该多大才最为合适。这里讲的单元究竟是小到一个函数，大到一个类，还是一个相关的类的集合？事实上，这些有关单元的例举都是正确的，单元的大小和进行单元测试的目的、粒度要求等有直接关系。GB/T 8567：2006 对软件单元做了明确定义。

软件单元：CSCI(Computer Software Configuration Item)设计中的一个基本单元可以是 CSCI 的一个主要分支中的一个组成部分、一个类、对象、模块、函数、子程序或者数据库。软件配置项可以出现在层次结构的不同层上并可以由其他的软件配置项组成。设计中的软件配置项与实现它们的代码和数据实体(例程、过程、数据库、数据文件等)与包含这些实体的计算机文件之间不一定有一一对应的关系。

从这个定义，可以很明确地看到单元的大小是不固定的，可以是函数，也可以是类、对象，甚至是一个模块或子程序。这个可以消除我们有关单元定义的疑惑，但是细心的人可能很快又会产生一个新的疑问，那就是一个函数、一个类是不可能单独运行的，动态测试如何来做呢？

单元测试因为其测试对象无法单独运行，一般在进行单元测试时需要编写一些辅助单元（模块），这些辅助单元（模块）和被测单元（模块）共同构成一个可以动态运行的测试环境（如图 10-16 所示）。

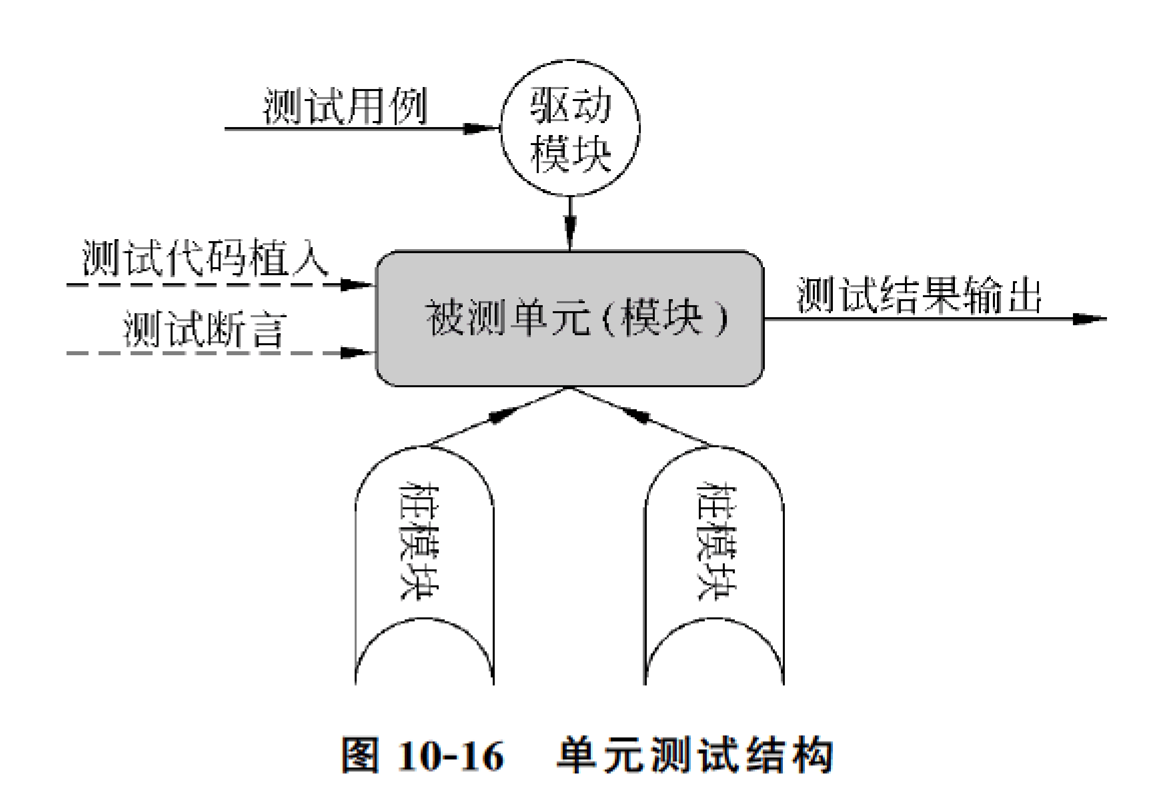


图 10-16　单元测试结构

通常，辅助单元按照其在测试环境中的作用可以分为驱动模块和桩模块，定义如下：

- 驱动模块：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果。
- 桩模块：用以代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做。

2．测试内容和方法

广义上说，单元测试的目的主要是为了确认被测模块的内外部行为，这里有两层含义，一个是模块内部需要验证模块的源代码编写结构符合设计规格要求，其处理流程符合业务需求规格，这是基本要求。另外，对于模块代码的编写质量应该也是单元测试的一个关注点，例如死代码（因程序员编写逻辑错误，使得某条件永远不可能被触发而导致永远不会得到执行的一部分代码）、冗余代码（重复代码是冗余代码的一个表现，如某个字符串处理的函数反复在多个类中出现，这个函数代码完全可以单独抽取出来作为一个公共类库使用，从而节约代码，达到最大程度的复用）。这是单元测试的第一层含义，单元测试的另一层含义则是模块的外部行为，这是单元测试的一个重要层面，无论内部代码处理的多么精致，但是模块所实现的功能是违反业务需求的，则这个模块就是不合格的。

对于模块的内部行为测试，一般从以下几个角度着手：

① 路径测试。路径测试的用例来源一般是模块的设计规格说明，参照其中的控制流程图，设计足够的路径组合，为每一条路径准备合理的测试数据以进行测试。路径测试也称为路径覆盖，针对路径测试的度量称之为路径覆盖度。通常，一个模块的路径组合是很

多的,完全的路径覆盖是不可能的,因此在对模块进行路径测试时一般只对基本路径进行测试,所谓基本路径指在控制流图中每一条路径至少会引入 1 条新边的逻辑路径。关于路径测试可参考 10.2.2 节的有关内容。

② 错误处理测试。检查模块的错误处理功能是否包含有错误。例如,正常、错误提示信息描述是否清晰准确,易于理解;非法数据的检查;是否出错原因报告有误、是否对错误条件的处理有误;在对错误处理之前错误条件是否已经引起系统的干预等。

③ 边界测试。对于控制流中存在判定条件,要特别注意对于条件值的以下情况进行测试,即刚好等于、大于或小于确定的比较值;要注意对于数据流中的有要求的数据进行边界的临界值测试,这些边界可能是数据长度、数据类型等,如对于文件路径这样的字符串输入,要测试的数值范围是小于等于 255 个字符的字符串,但是空字符串则应是被禁止的。

对于模块的外部行为的测试主要通过以下 3 个方面的测试来实现:

① 模块接口测试。模块接口测试主要是对通过模块进行处理或者传递的双向数据流(流经模块的数据分为输入和输出两个类别)进行测试。主要的模块类别有以下几类:参数表、调用子模块的参数表、全局数据、文件输入/输出操作。

② 局部数据结构测试。局部数据结构测试主要检查数据类型声明、初始化、默认值等方面的问题,还要检查验证全局数据对模块的影响。

③ 性能测试。随着软件应用规模的变化,性能测试愈来愈多地得到重视。系统的整体性能瓶颈最终还是要定位到某个模块上,因此模块级的性能测试成了一个新的研究热点。这类测试通常采用自动化测试工具实现,通过对关键路径的测试以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对于性能评价是十分有用的。

通常单元测试在源程序代码编制完成,经过评审和验证,确认没有语法错误之后进行。单元测试也要设计测试用例,利用设计文档,设计可以验证程序功能、找出程序错误的一组测试用例(又称做测试套件)。对于每一组测试输入,应有预期的正确结果。

进行单元测试,主要的方法是白盒测试法,严格来说静态测试中的代码检查和审查也是单元测试的方法之一,这对于确认模块的内部行为是十分有效的。目前,业界流行的主流编程语言基本上都有一套实践总结出来的编码规范,编码规范也获得了很多企业和个人的认可,有着较为广泛的应用。代码规范往往作为代码检查/审查的依据,用以发现存在于源代码中的不规范的语法使用或者开发人员个体的不良编码习惯。表 10-13 列出了是有关 Java 编码规范的一个片段。

表 10-13　Java 代码检查实例

序号	测试项	测 试 内 容	质量保证标准
1	下标	是否有下标变量越界错误	健壮性
2	除数	是否包含有除零错误的可能	健壮性
3	Get 方法	当对一个不知是否为空的对象取其属性值会引起空指针异常。如果空指针异常没有被接收程序将终止。例如:BusinessData1.getBusinessDate2.getOid()当 BusinessData1.getBusinessDate2 为 null 时,BusinessData1.getBusinessDate2.getOid()将发生异常	健壮性

续表

序号	测试项	测 试 内 容	质量保证标准
4	字符串	在字符串比较和将字符串写入数据表前应 Trim()掉它的前后空格	健壮性
5	字符串连接符"+"	将字符串连接操作中的+操作符同加法运算中的+操作混淆将导致奇怪的结果。例如:y 为 int 类型,y 的值为 5,g. drawString("y+2="+y+2,30,30);将显示 y+2=52	正确性

静态分析也是单元测试的一个有效方法,目前有众多厂家致力于这方面的研究,比如 Rational 公司的 Purify、Telelogic 公司的 Logiscope 等都是这里面的佼佼者。

动态测试对于单元测试是不可或缺的,如果有过开发经验,那么大家一定有过这样的经历:在编译完成后,运行程序时出现了异常错误,为了跟踪这个错误,定位引起错误的代码行,要不断地调试。为了方便我们会植入一些输出的提示信息,其实这个过程在某种意义上可以说就是一次轻量级的动态单元测试。

目前,动态的单元测试以成熟的测试框架为发展方向,在这个领域以 xUnit 为代表,包括 JUnit、DUnit、CppUnit 等针对不同语言的多个框架。受益于 Java 语言的流行程度,在很多书籍中对 JUnit 有较多的论述,本书对这部分内容不再细述。

3. 单元测试案例剖析

1) 测试需求分析

单元测试的测试需求主要源自于系统设计说明书,通过对系统设计说明书的分析找出单元测试的基本测试点。那么对于本例,系统是在 VC++ 集成开发环境中开发的,采用了面向对象的设计开发技术。因此,在进行该系统的单元测试时,需要从面向对象的角度对测试单元进行分析,比如继承性测试、类方法的测试等。本书的重点不是介绍面向对象的软件测试方法,因此对于使用的测试技术不做深入解析,而只是结合案例从一般的单元测试的角度诠释测试的设计与实现。

从本案的系统设计说明书中可以看到,系统从 UI 接收的数据保存至数据库的过程被抽象封装为独立的逻辑处理类,例如 CMClerk 是员工信息管理的逻辑处理类、CMPact 是员工合同的处理类等。为了说明方便,下面均以 CMClerk 为例作为单元测试对象。

单元测试的最终目的有两个,一个是外部行为的考察,一个是内部行为的考察,但是我们认为内部行为的考察本质上源于外部行为考察的需要。因为,一个单元的内部构造无论多么精巧,但如果它所提供的使用接口是完全错误的,则这个单元的测试就不能说是通过的,也就是说无法进入系统的集成。

在本案中,关于员工信息管理的业务需求对单元 CMClerk 的外部行为做出了规约(如图 10-17 所示),这构成了 CMClerk 的基本测试需求:

(1) 保存员工信息。

员工信息包含多项内容,其中部分数据是选项数据,如性别只能取“男”和“女”这两个数据;婚姻情况只能取“未婚”、“已婚”和“离异”三类数据。那么对于非取值范围内的数据是不是 CMClerk 单元测试的测试项呢?我们知道,CMClerk 是中间数据逻辑处理类,数据接收是在另外的 UI 类中,这里面有两种处理方式,一种是 UI 对非法数据进行过滤,一

种是 CMClerk 对非法数据进行过滤，经对系统设计进一步分析和与相关开发人员的确认，本案采取的是第一种处理方式，因此在 CMClerk 的单元测试中，这两类非法数据不作为测试项。同样，对于其他未作出说明的数据项也不进行非法输入测试。

数据项身份证号有几种情况，一个是位数，15 位或者 18 位，一个是末尾含有字母的和全部为数字的。这几种情况要求 CMClerk 都能够处理，需要对这几类数据进行测试。

数据项出生日期要求满足日期格式，另外还有一个潜在的要求即出生日期距离录入日期应不小于 18 岁(非关键性隐含需求)。

员工信息在保存时要通过职业状态判断员工是否是本单位曾辞退员工，禁止招录被辞退员工。

(2) 员工各类职业状态的处理。

(3) 修改员工基本信息。

(4) 删除员工基本信息。

(5) 员工信息重复性检查。

【案例项目 HRMIS 员工信息重复性检查】

配置项标识		需求覆盖情况描述
P01_YGXX	P01_01_JBXX	• 保存员工信息。人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，主要职员信息如下： 姓名、性别、出生日期、政治面貌、文化水平、婚姻情况、家庭住址、身份证号、办公电话、移动电话、紧急情况下的联系人和联系方式、毕业院校、入职时间、岗位及职责 其中，性别包含男、女两个类别；婚姻情况包括未婚、已婚、离异三种情况。 • 员工各类职业状态的处理 • 修改员工信息 • 删除员工基本信息 • 员工信息重复性检查

从本案的系统设计说明书中，可以看到关于 CMClerk 的功能接口有 9 个，它们是 AddClerk(增加员工信息)、IsExist(员工是否已存在)、ModifyClerkInfo(修改员工信息)、RemoveClerk(删除员工信息)、SetClerkSex(修改员工性别)和 SetClerkState(修改员工状态)、AddChangeEvent(增加变更记录)、AddTrainExe(增加培训记录)和 HadTrain(指定员工是否已参加过该项培训)(类图表示如图 10-17 所示)。

CMClerk 的这几个方法的作用及其参数含义，可以从“人力资源管理系统设计说明书”中获得，如表 10-14 所示。

CMClerk	
+AddChangeEvent ( )	: Boolean
+AddTrainExe ( )	: Boolean
+AddClerk ( )	: Boolean
+HadTrain ( )	: Boolean
+IsExist ( )	: Boolean
+ModifyClerkInfo ( )	: Boolean
+RemoveClerk( )	: Boolean
+SetClerkSex ( )	: Boolean
+SetClerkState( )	: Boolean

图 10-17 CMClerk

表 10-14　CMClerk 方法解释

1	AddClerk(CString ygName/ * 姓名 * /, CString ygMale/ * 性别 * /, COleDateTime ygBirth/ * 出生日期 * /, CString ygPolity/ * 政治面貌 * /, CString ygCulture/ * 文化程度 * /, CString ygMarry/ * 婚姻状况 * /, CString ygAdr/ * 家庭住址 * /, CString ygPriCode/ * 身份证 * /, CString ygPhone/ * 办公电话 * /, CString ygMobile/ * 移动电话 * /, CString ygCallWho/ * 紧急联系人 * /, CString ygCallNo/ * 联系方式 * /,CString ygGraduate/ * 毕业院校 * /, COleDateTime ygInTime/ * 入职时间 * /, CString ygRole/ * 岗位 * /,CString ygDuty/ * 职责 * /, CString ygState/ * 职业状态 * /,CString ygNation/ * 民族 * /,CString ygDepart/ * 部门 * /)	保存员工信息
2	IsExist(CString cPriCode) 参数:cPriCode-身份证号	检查员工是否已存在
3	ModifyClerkInfo(CString ygCode,CString ygModified,int mPointer) 参数：ygModified-修改内容 mPointer-修改项指示器	修改员工信息
4	RemoveClerk(CString strCode,CString ygName, CString ygPriCode)	删除员工信息
5	SetClerkSex(CString ygCode, CString sNewSex)	设置员工性别
6	SetClerkState(CString ygCode, CString sNewState)	设置员工职业状态
⋮	⋮	

2）语句覆盖

```
BOOL CMClerk::AddClerk(CString ygName/ * 姓名 * /, … ,CString ygDepart/ * 部门 * /)
{
    / *
        新增一个用户
    * /
    CString strSQL;
    CString uTemp;
    //Exist?
    int iState=IsExist(ygPriCode);
    if(iState==0)
    {
        CADORecordset * pRs=new CADORecordset(theApp.pHr_AdoDb);
        pRs->Open("T_YG", CADORecordset::openTable);
        pRs->AddNew();
        pRs->SetFieldValue("YGXM",ygName);
        pRs->SetFieldValue("YGXB",ygMale);
        pRs->SetFieldValue("YGSHR",ygBirth);
        pRs->SetFieldValue("ZHZHMM",ygPolity);
        pRs->SetFieldValue("WHSP",ygCulture);
        pRs->SetFieldValue("HYZHK",ygMarry);
        pRs->SetFieldValue("JTZHZH",ygAdr);
        pRs->SetFieldValue("SHFZH",ygPriCode);
        pRs->SetFieldValue("BGDH",ygPhone);
```

```
        pRs-> SetFieldValue("YDDH",ygMobile);
        pRs-> SetFieldValue("JJLXR",ygCallWho);
        pRs-> SetFieldValue("JJLXDH",ygCallNo);
        pRs-> SetFieldValue("BYYX",ygGraduate);
        pRs-> SetFieldValue("RZHSHJ",ygInTime);
        pRs-> SetFieldValue("GZGW",ygRole);
        pRs-> SetFieldValue("ZHZ",ygDuty);
        pRs-> SetFieldValue("ZYZT",ygState);
        pRs-> SetFieldValue("YGMZ",ygNation);
        pRs-> SetFieldValue("BMBH",ygDepart);

        pRs-> Update();
        pRs-> Close();

        delete pRs;

        return 1;
    }
    else
        return-1;                    //用户已存在
}
```

先来看 CMClerk 的第一个方法 AddClerk，根据函数代码可以做出控制流程图(如图 10-18 所示)以方便对函数结构进行分析。这个函数总体上比较简单，只包含一个判定条件，从流程图上可以看出，只需沿着图 10-18 所示深浅两条标注线进行测试，就可以使函数内的所有语句都执行一遍，也就是达到语句覆盖要求。而要使函数按照这样两条路径运行，只需使判定条件取一次真和取一次假即可。基于这个分析，针对深色线标注路径可以构造出以下用例。

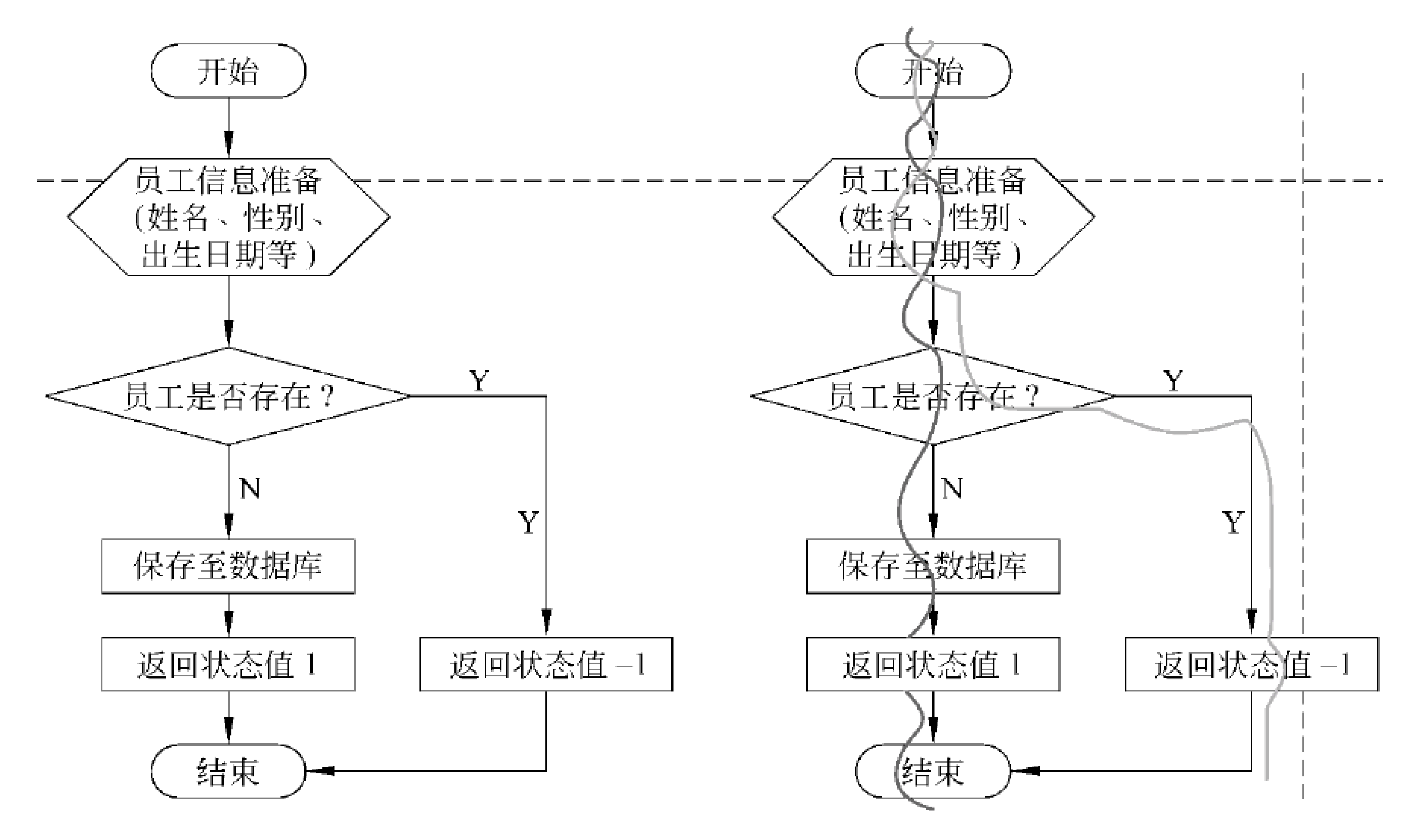


图 10-18　AddClerk 控制流程图

【用例 10-1】

用例名称	测试 AddClerk	用例标识	P200901UT-YGGL-YGLR-01
测试追踪	业务需求说明书:人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中,TR01		
用例说明	测试员工信息管理类保存员工信息(AddClerk)方法,对于给定的合法数据能否正常保存		

用例的初始化	硬件配置	无特殊要求
	软件配置	软件可以正常启动运行,正常连接 MySQL 数据库
	测试配置	无特殊要求
	参数设置	部门组织结构已初始化

操作过程

序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号(身份证号要满足前置条件)等)	—	—	参见桩数据
2	将桩模块数据传递到 AddClerk 中	—	—	
3	调用方法 AddClerk	—	—	
4	查询数据库,检验数据保存是否正确	员工信息成功保存	可以查询到新录入的员工信息	
5	查看函数返回值是否为 1			

前提和约束	员工身份证号在已保存的员工信息中已存在		
过程终止条件	无		
结果评价标准	身份证号为 370205198612301234 的员工信息保存成功,函数返回 1		
设计人员	—	设计日期	—

桩数据如下:

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "370205198612301234"
CString strYGSex="男";
COleVariant oleBirth("1986-12-30");
COleDateTime dtBirthDay=oleBirth;
CString strYGPolity="民主联盟";
CString strYGCulture="硕士";
CString strYGMarry="已婚";
CString strYGHome="青岛市环海路 465 号";
CString strYGPhone="86261283";
CString strYGMobile="18978273909";
```

```
CString strCallWho="林轩";
CString strCallNo="13628390987";
CString strYGGradu="中国农业大学信息工程学院";
COleVariant oleInTime("1986-12-30");
COleDateTime dtInTime=oleInTime;
CString strYGRole="评测工程师";
CString strYGState="试用";
CString strYGNation="哈萨克族";
CString strYGDepart="1";            //与合同的测试用例要一致
CString strYGDuty="负责应用软件的测试、缺陷记录及报告编写";
```

对于浅色标注路径，根据前述分析我们知道当判定条件取值为假时函数就按照浅色执行，因此可以在用例 10-1 的基础上很容易就可以构造出用例 10-2，即再执行一遍用例一即可达到测试目的。

【用例 10-2】

用例名称	测试 AddClerk		用例标识	P200901UT-YGGL-YGLR-02
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	测试员工信息管理类保存员工信息(AddClerk)方法，数据重复时保存失败			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号(身份证号要满足前置条件)等)	—	—	桩数据同用例 P200901UT-YGGL-YGLR-01
2	将桩模块数据传递到 AddClerk 中	—	—	
3	调用方法 AddClerk	—	数据保存失败	
4	查看函数返回值是否为－1		函数返回－1	
前提和约束	① 员工身份证号在已保存的员工信息中已存在 ② P200901UT-YGGL-YGLR-01 通过			
过程终止条件	无			
结果评价标准	身份证号为 370205198612301234 的员工信息保存失败，函数返回－1			
设计人员	—	设计日期	—	

3）路径覆盖

我们知道单纯的语句覆盖是达不到测试要求的，为了进一步对 CMClerk 的 AddClerk 方法进行测试，来分析一下这个函数的基本路径。

前面已经根据 AddClerk 的源代码给出了控制流程图（如图 10-19 左边区域所示），根据这个流程图我们可以很容易地导出其控制流图（如图 10-19 右边区域所示），根据计算公式 V(G)＝m－e＋2＝4－4＋2＝2，计算出该段程序的基本路径有两条，即 0—1—2—3 和 0—1—3。这个结果和之前语句覆盖的分析是一致的，但是这并不意味着语句覆盖可以代替路径覆盖，这只是程序本身特点造成的，而正相反路径覆盖的一个基本要求就是保证程序中每条语句至少执行一次，也就是说路径覆盖可以达到语句覆盖的目的。

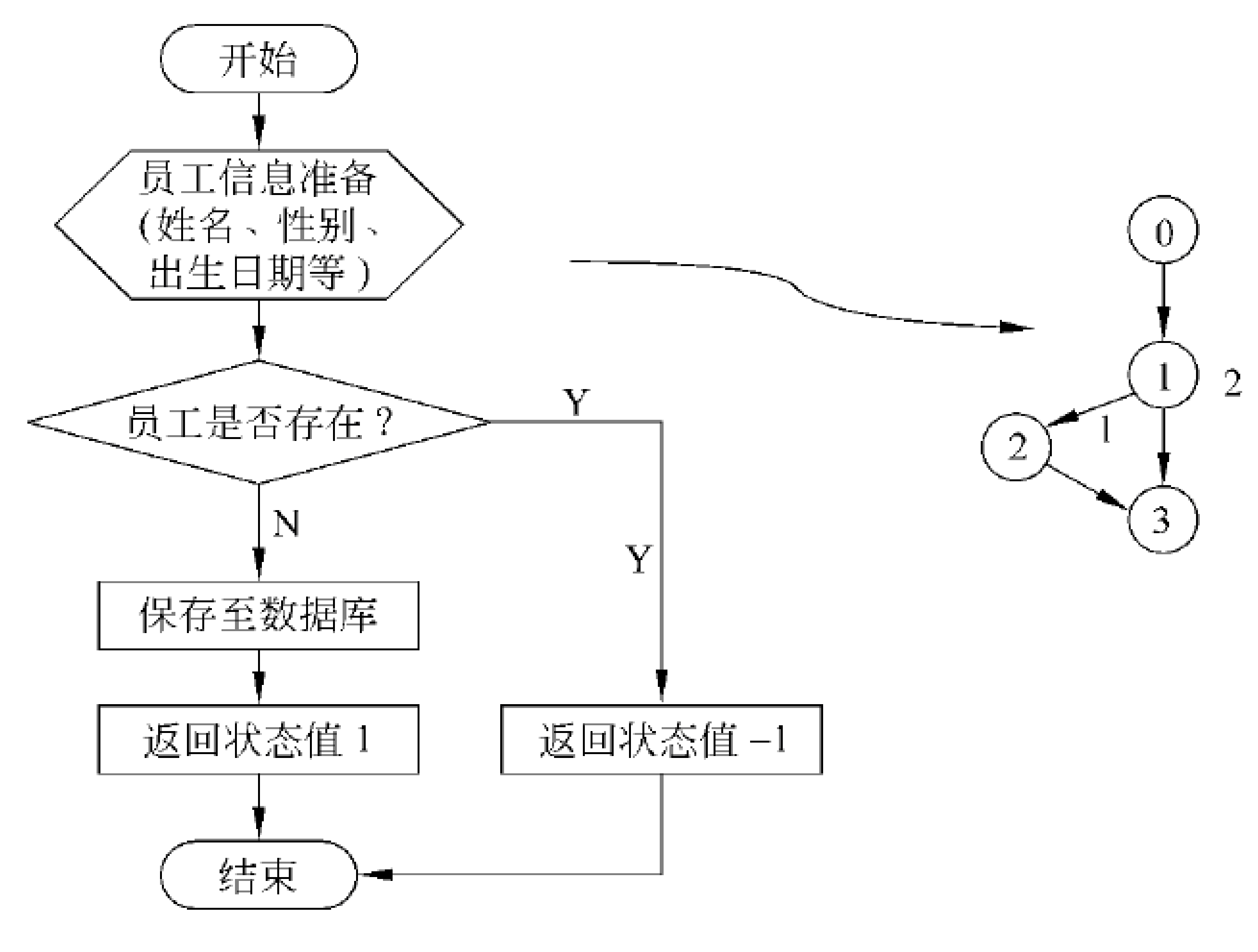


图 10-19　AddClerk 控制流图

4）边界测试

根据前面的分析知道在 CMClerk 这个方法中处理的数据具有比较明确的边界的是身份证号和移动电话号码。

移动电话号码由 11 位数字组成，所有位数超过 11 位或者不足 11 位的移动通信号码都是不合法的；身份证号只能是 15 位或者 18 位的，小于 15 位、超过 18 位或者介于 15 位和 18 位之间的都是非法身份证号（如图 10-20 所示）。但是，因为选定的测试对象是中间的数据逻辑处理类 CMClerk，假定 CMClerk 的接收数据都是合法的，也就是不对这类边界进行测试。那么剩下来还有一个边界，那就是身份证号，因为身份证号有含有字母的，这是一个隐含的边界，要求 CMClerk 能够处理这两类身份证号。基于这个边界分析结果，需要在上述用例的基础上补充以下用例。

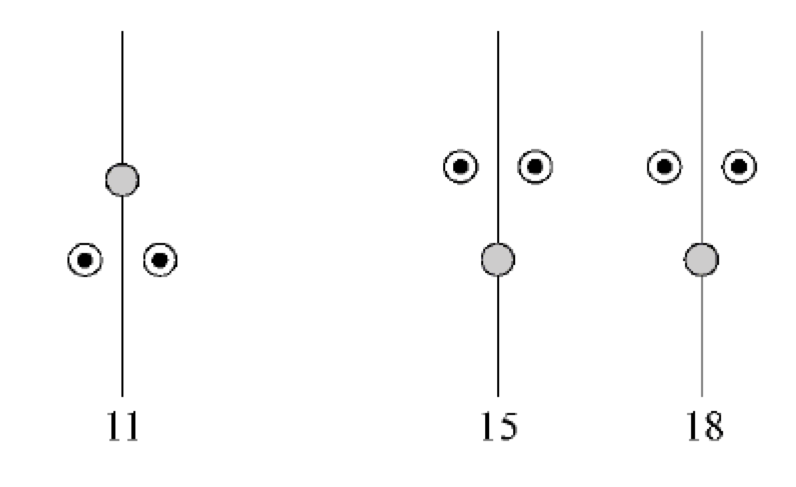


图 10-20　电话号码和身份证号边界分析

【用例 10-3】

用例名称	测试 AddClerk	用例标识	P200901UT-YGGL-YGLR-03
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	测试员工信息管理类保存员工信息(AddClerk)方法，测试含有字母的身份证号码的处理情况		
用例的初始化	硬件配置	无特殊要求	
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库	
	测试配置	无特殊要求	
	参数设置	部门组织结构已初始化	

操作过程

序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号(身份证号要满足前置条件且末尾为字母)等)	—	—	参见桩数据
2	将桩模块数据传递到 AddClerk 中	—	—	
3	调用方法 AddClerk	—	—	
4	查询数据库，检验数据保存是否正确	员工信息成功保存	可以查询到新录入的员工信息	
5	查看函数返回值是否为 1		函数返回 1	
前提和约束		—		
过程终止条件		无		
结果评价标准		身份证号为 37020519861230123X 的员工信息保存成功，函数返回 1		
设计人员		—	设计日期	—

桩数据如下：

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "37020519861230123X"
CString strYGSex="男";
COleVariant oleBirth("1986-12-30");
COleDateTime dtBirthDay=oleBirth;
CString strYGPolity="民主联盟";
CString strYGCulture="硕士";
CString strYGMarry="已婚";
CString strYGHome="青岛市环海路 465 号";
CString strYGPhone="86261283";
```

```
CString strYGMobile="18978273909";
CString strCallWho="林轩";
CString strCallNo="13628390987";
CString strYGGradu="中国农业大学信息工程学院";
COleVariant oleInTime("1986-12-30");
COleDateTime dtInTime=oleInTime;
CString strYGRole="评测工程师";
CString strYGState="试用";
CString strYGNation="哈萨克族";
CString strYGDepart="1";            //与合同的测试用例要一致
CString strYGDuty="负责应用软件的测试、缺陷记录及报告编写";
```

查看 AddClerk 的函数体可以看到 AddClerk 调用了函数 IsExist，因此要对 AddClerk 做一个真正的白盒测试，需要进一步分析 IsExist 的函数体（如下代码所示），可以看到在本函数中判断为员工存在的条件是身份证和职业状态，其中职业状态有“实习”、“试用”、“正式”、“返聘”和“辞退”这 5 类。通过解读这些信息，可以更容易地为 AddClerk 的测试用例准备测试数据。

```
BOOL CMClerk::IsExist(CString cPriCode)
{
    /*
      员工信息是否已经存在
      0—不存在
      1—已存在
    */
    CString strFilter;
    CString strSQL;
    strSQL.Format("Select *  From T_YG Where SHFZH='%s' And ZYZT in ('实习','试用',
    '正式','返聘','辞退')",cPriCode);

    CADORecordset* pRs=new CADORecordset(theApp.pHr_AdoDb);
    if(pRs->Open((LPCTSTR)strSQL))
    {
        int num=pRs->GetRecordCount();
        if (num<=0)
            return 0;
        else
        {
            return 1;
        }
    }
```

```
    return 0;
}
```

类 CMClerk 的方法 IsExist 的结构也比较简单，按照前面的分析设计方法，可以构造出两个测试用例分别使 IsExist 取一次真和取一次假，从而使这个测试达到路径覆盖的目的。

【用例 10-4】

<table>
<tr><td>用例名称</td><td colspan="2">测试 IsExist</td><td>用例标识</td><td>P200901UT-YGGL-YGLR-04</td></tr>
<tr><td>测试追踪</td><td colspan="4">业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01</td></tr>
<tr><td>用例说明</td><td colspan="4">测试员工信息管理类检验指定员工是否存在的(IsExist)方法，当员工为新员工时函数是否返回 0</td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="3">无特殊要求</td></tr>
<tr><td>软件配置</td><td colspan="3">软件可以正常启动运行，正常连接 MySQL 数据库</td></tr>
<tr><td>测试配置</td><td colspan="3">无特殊要求</td></tr>
<tr><td>参数设置</td><td colspan="3">部门组织结构已初始化</td></tr>
<tr><td colspan="5">操作过程</td></tr>
<tr><td>序号</td><td>输入及操作说明</td><td>期望的测试结果</td><td>评价标准</td><td>备　注</td></tr>
<tr><td>1</td><td>在桩模块中构造一组员工信息的测试数据（包括姓名、性别、年龄、出生日期、身份证号等）</td><td>—</td><td>—</td><td>参见桩数据</td></tr>
<tr><td>2</td><td>将桩模块数据传递到 IsExist 中</td><td>—</td><td>—</td><td></td></tr>
<tr><td>3</td><td>调用方法 IsExist</td><td>—</td><td>—</td><td></td></tr>
<tr><td>4</td><td>查看函数返回值是否为 0</td><td>—</td><td>函数返回值为 0</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">前提和约束</td><td colspan="3">员工身份证号在已保存的员工信息中不存在</td></tr>
<tr><td colspan="2">过程终止条件</td><td colspan="3">无</td></tr>
<tr><td colspan="2">结果评价标准</td><td colspan="3">身份证号为 370205198612301234 的员工信息不存在，函数返回 0</td></tr>
<tr><td colspan="2">设计人员</td><td>—</td><td>设计日期</td><td>—</td></tr>
</table>

桩数据如下：

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "370205198612301234"
```

【用例 10-5】

用例名称	测试 IsExist		用例标识	P200901UT-YGGL-YGLR-05
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	测试员工信息管理类检验指定员工是否存在的(IsExist)方法，当员工信息存在时，函数是否返回 1			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号等)	—	—	参见桩数据
2	将桩模块数据传递到 IsExist 中	—	—	
3	调用方法 IsExist	—	—	
4	查看函数返回值是否为 1	—	函数返回值为 1	
前提和约束	从职业状态为“实习”的员工信息中抽取一个员工身份证号			
过程终止条件	无			
结果评价标准	身份证号为 370205198612301234 的员工信息已存在，函数返回 1			
设计人员	—	设计日期	—	

桩数据如下：

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "370205198612301234"(假设)
```

另外，因为职业状态为“退休”的职工有一定特殊性，增加一个边界测试用例。

【用例 10-6】

用例名称	测试 IsExist	用例标识	P200901UT-YGGL-YGLR-06
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	测试员工信息管理类检验指定员工是否存在的(IsExist)方法，当员工职业状态为“退休”时，函数是否返回为 0		

续表

用例的初始化	硬件配置	无特殊要求			
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库			
	测试配置	无特殊要求			
	参数设置	部门组织结构已初始化			
操作过程					
序号	输入及操作说明		期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据（包括姓名、性别、年龄、出生日期、身份证号等）		—	—	参见桩数据
2	将桩模块数据传递到 IsExist 中		—	—	
3	调用方法 IsExist		—	—	
4	查看函数返回值是否为 0		—	函数返回值为 0	
前提和约束			员工身份证号在已保存的员工信息中存在，其职业状态为“退休”		
过程终止条件			无		
结果评价标准			身份证号为 370205198612301234 的员工信息不存在，函数返回 0		
设计人员			—	设计日期	—

桩数据如下：

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "370205198612301234"
```

一个类是无法独立运行的，所以单独对一个类进行测试是比较困难的，为了测试一个类模块一般需要编写驱动模块，通过驱动模块调用被测类(Class Under Testing，CUT)的方法进行测试。如果要测试的类是非底层类，那么可能还需要编写一些桩模块，桩模块为 CUT 提供底层数据，和驱动模块共同完成被测类的测试。

单元测试是软件测试领域无论是开发人员还是测试人员都倾注了很大热情的领域，发展迅速，自动化程度相对较高，目前主流的开发平台都集成了一些辅助进行单元测试的工具。在开源社区，也有很多类似的项目在单元测试领域进行探索和实践，也取得了一些足以可以使用的单元测试集成框架。结合本书的案例项目，拟在下一节给出针对 C++ 平台的单元测试框架 CppUnit 的使用实例。

4. 应用 CPPUnit 编写单元测试用例

1) xUnit 家族与 CppUnit

xUnit 是从 Junit 发展而来的。起初，Kent Beck 和 Eric Gamma 写了一组 Java 类，用

以实现单元测试的自动化,他们将这组类库称之为 JUnit。JUnit 因其先进的测试原理和结构很快引来更多的人加入研究行列,并陆续发展出针对其他多种开发语言的版本,比如针对 C/C++ 的 CUnit/CppUnit,针对 Delphi 的 DUnit,针对 Visual Basic 的 VBUnit,针对.NET 平台的 NUnit 等。

这些不同版本的测试框架使用相近的规则,只要注意很少的一些语法差异,xUnit 的使用者就能很快从一个版本迁移到另一个版本。

	Junit	CppUnit	NUnit	DUnit	VBUnit
适用语言	Java	C++	.Net	Delphi	VisualBasic
开发工具	Java	C++	C#	—	—

CppUnit 是 xUnit 系列中的 C++ 实现版本,它是从 JUnit 移植过来的,第一个移植版本由 Michael Feathers 完成,CppUnit 遵循 GNU LGPL(Lesser General Public License),可以免费获取源码和研究使用。

如果说测试发展的目标是 A-TRIP(自动化、彻底的、可以重复的、独立的、专业的),那么在单元测试领域,xUnit 的出现可以说向这个方向迈进了一步。

2) CppUnit 应用详解

(1) CppUnit 的构成。作为一个完整的 CppUnit framework,虽然源码所在的实际路径可能不尽相关,但从逻辑上讲它们被划为如下几个部分:

- core:CppUnit 的核心部分;
- output:掌管结果输出;
- helper:一些辅助类;
- extension:作为单元测试的延伸,对 CppUnit core 部分的扩展(如常规测试、重复测试);
- listener:监视测试进程和测试结果;
- textui:一个运行单元测试的文本环境;
- portability:提供针对不同平台的移植设置。

(2) TestCase/TestSuite/TestFixture,如图 10-21 所示。

TestCase:测试用例,在 CppUnit 中一般继承自 CppUnit::TestCase。

TestSuite:一组相互关联的测试用例,称之为一个测试包,也就是 TestSuite。

TestFixture:用于包装测试类使之具有 setUp 方法和 tearDown 方法。利用它,可以为一组相关的测试提供运行所需的公用环境(即所谓的 Fixture(夹具))

(3) failure 和 error。CppUnit 中有两种类型的错误,它们分别是:failure 和 error。一个 failure 是可预期的,并可以为断言(assert)所侦测到;而 error 则是不可预期的,由异常标示,它并非框架代码所产生。

(4) 使用 CppUnit 进行单元测试。

在没有单元测试工具之前,开发团队成员主要的单元测试手段是调试器,调试器当然

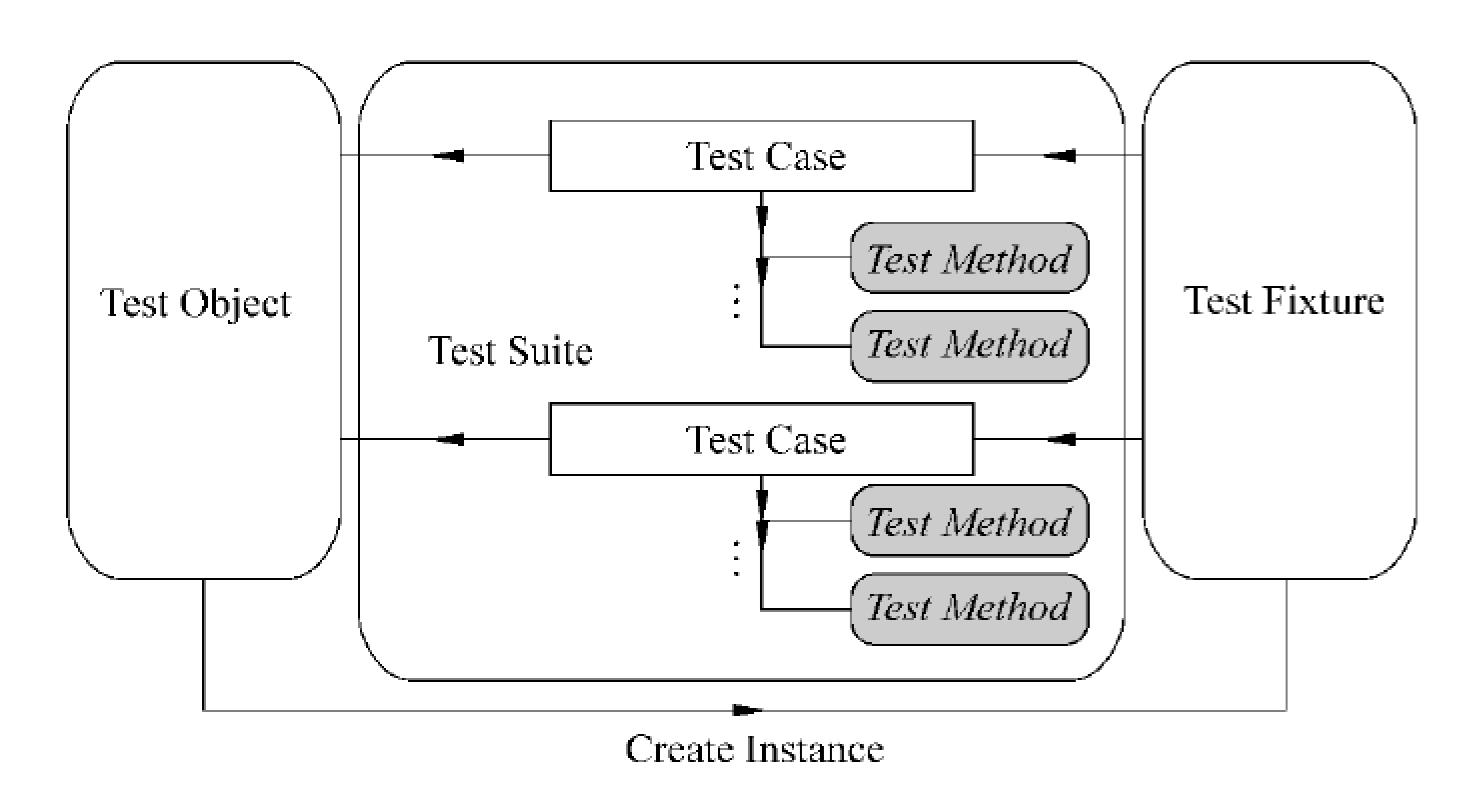


图 10-21　TestCase/TestSuite/TestFixture

可以满足在任何时候检查每个程序或者子程序的变量数值的要求，通过设置断点，也可以一步一步地对变量进行跟踪，验证变量数值的变化情况，判断其是否是期望的结果，但是这种工作方式的缺点是明显的，那就是低效，灵活性不足。

CppUnit 的出现挽救了 C++ 的单元测试，通过单元测试工具的引入和应用，开发人员或者测试人员只需编写少量的代码即可对模块展开测试。

关于 CppUnit 与 VC++ 集成的问题请参阅参考文档《环境配置说明》，假定 CppUnit 测试框架已经配置好，接下来将结合本书的案例项目直接讲述有关在 VC++ 集成开发环境中如何使用 CppUnit 进行单元测试的内容。

本例要通过调用 CppUnit 的可视化测试组件，因此在项目的主文件中（本例是 hrmis. cpp）引入头文件 TestRunner. h，该文件在 CppUnit 头文件库的位置是 cppunit/ui/mfc，也就是：

```
#include <cppunit/ui/mfc/TestRunner.h>
```

每一次测试的运行可能包含了许多测试实例，这些用例彼此间可能呈现层状结构，而每个测试实例的创建或者说实例化都是由某个与之对应的类工厂完成的。为了较好的管理这些类工厂，实现其生命周期的自动操控（产生、销毁及资源回收），CppUnit 采用注册机制来实现这个目标。

为了保证在测试执行时，测试用例能够有效地实例化，并完成自我管理，因此需要将我们编写的测试用例在类工厂注册，要注册用例，需要在 hrmis. cpp 中加入 TestRactoryRegistry. h 这个文件。

```
#include <cppunit/extensions/TestFactoryRegistry.h>
```

引入这两个头文件之后（如图 10-22 所示），就可以调用 CppUnit 的可视化操作界面进行测试了。

本例中，测试用例需要与数据库交互，一方面提交数据，另一方面从数据库中提取数据并与源数据进行比较，以验证操作的正确性。本例中的用例大多是以这种模式来设计的。为了调用 CppUnit 的可视化测试操作界面，需要在如图 10-23 所示位置，加入以下

代码：

```
CppUnit::MfcUi::TestRunner
runner;runner.addTest( CppUnit::TestFactoryRegistry::getRegistry().makeTest() );
```

```
#include "TestCaseReg.h"

#include <cppunit/ui/mfc/TestRunner.h>
#include <cppunit/extensions/TestFactoryRegistry.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// ChrmisApp
```

图 10-22　源码样本：引入头文件

```
if(pHr_AdoDb->Open((LPCTSTR)strConnection))
{
    //写入日志文件
    // AfxMessageBox("数据库以连接");
    pHr_AdoDb->Execute("Set Names GBK"); //MySql 显示中文

    /*单元测试，调用TestRunner*/
    // declare a test runner, fill it with our registered tests and run them
    CppUnit::MfcUi::TestRunner runner;

    runner.addTest( CppUnit::TestFactoryRegistry::getRegistry().makeTest() );

    runner.run();
}
else
    return false;
```

图 10-23　CppUnit 可视化测试界面调用代码

这样有关进行单元测试的准备工作已经基本完成，应用 CppUnit 进行单元测试的脚手架已经构建好，接下来就是 TestSuite（测试套件）和 TestCase（测试用例）的设计开发了。

有关测试套件和测试用例的概念之前已经论述，下面着重看一下测试套件的构建宏 CPPUNIT_TEST_SUITE。CPPUNIT_TEST_SUITE 实现了测试套件的创建、销毁和用例的封装，提供了以下三个功能函数。

- CPPUNIT_TEST_SUITE()：创建一个 TestSuite；
- CPPUNIT_TEST()：添加 TestCase，可以添加多个，事实上测试套件正是对多个测试用例的封装；
- CPPUNIT_TEST_SUITE_END()：结束创建 TestSuite。

CppUnit 提供了 TestCase 这样一个测试用例基类，我们在编写测试用例之前首先需要新构建一个派生于 TestCase 的一个新类，类的标识可以与被测试类的标识一致，而以 TestCase 结束，这样容易识别和管理（如图 10-24 左边所示）。在新产生的用例头文件中（本例中是 MClerkTestCase. h 文件）加入 TestCase. h 和 HelperMacros. h 的引用（如图 10-24 右边所示），否则编译不通过。

```
#include <cppunit/TestCase.h>
#include <cppunit/extensions/HelperMacros.h>
```

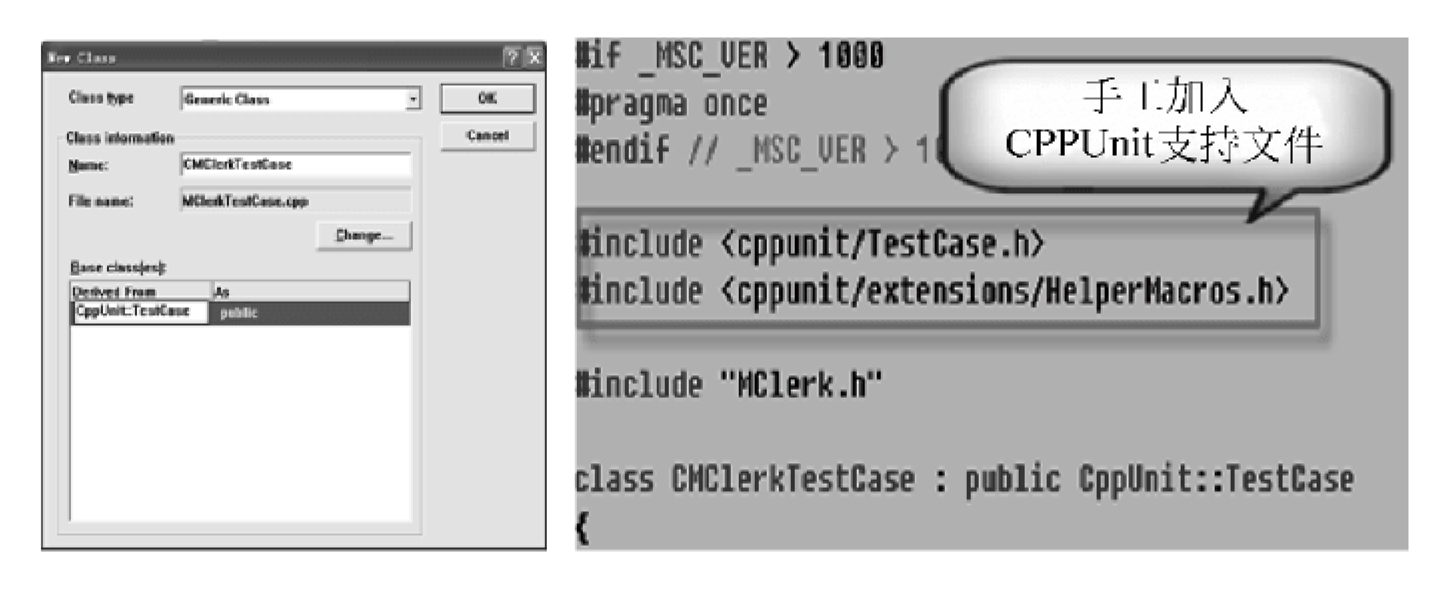


图 10-24 测试用例定义

在本例中，要测试的对象类是 CMClerk，CMClerk 在本例中主要用以员工信息的管理，比如新进员工信息的保存、原有员工信息的修改等内容。为了测试该类的这些方法，拟设计开发以下 5 个用例。

- AddClerk：新增员工记录；
- IsExist：员工是否已经存在；
- SetClerkSex：员工性别变更；
- SetClerkState：员工职业状态变更；
- RemoveClerk：员工信息删除。

这 5 个用例都是用来测试 CMClerk 的，因此它们是相互关联的，为了方便管理把这 5 个用例组织为一个测试套件，测试套件的定义和使用应用前面已有介绍。

CPPUNIT_TEST_SUITE()宏。CPPUNIT_TEST_SUITE()在用例类的头文件中编写：

```
//测试用例执行时将按照注册顺序进行
CPPUNIT_TEST_SUITE(CMClerkTestCase);
    CPPUNIT_TEST(AddClerk);
    CPPUNIT_TEST(IsExist);
    CPPUNIT_TEST(SetClerkSex);
    CPPUNIT_TEST(SetClerkState);
    CPPUNIT_TEST(RemoveClerk);
CPPUNIT_TEST_SUITE_END();
```

在设计测试用例时，尽量保持用例的方法与被测对象的方法一致，以方便识别(如图 10-25 所示)。

具体用例的编写，也就是测试类的方法的代码编写与一般的 VC++ 程序开发没有任何差异，因为这本书是软件测试类的书籍，对于软件开发的部分本书不加解释，在此直接给出编码实现后的用例，供读者参考。

图 10-25　测试接口定义

AddClerk

```
void CMClerkTestCase::AddClerk()
{
    CString strYGSex="男";

    COleVariant  oleBirth("1986-12-30");
    COleDateTime dtBirthDay=oleBirth;

    CString strYGPolity="民主联盟";
    CString strYGCulture="硕士";
    CString strYGMarry="已婚";
    CString strYGHome="青岛市环海路 465 号";
    CString strYGPhone="86261283";
    CString strYGMobile="18978273909";
    CString strCallWho="林轩";
    CString strCallNo="13628390987";
    CString strYGGradu="中国农业大学信息工程学院";

    COleVariant  oleInTime("1986-12-30");
    COleDateTime dtInTime=oleInTime;

    CString strYGRole="评测工程师";
    CString strYGState="试用";
    CString strYGNation="哈萨克族";
    CString strYGDepart="1"; //与合同的测试用例要一致
    CString strYGDuty="负责应用软件的测试、缺陷记录及报告编写";

     BOOL bVal = clerkFixture.AddClerk (strYGName, strYGSex, dtBirthDay, strYGPolity,
     strYGCulture,strYGMarry,
```

```
    strYGHome,strYGPriCode,strYGPhone,strYGMobile,strCallWho,strCallNo,strYGGradu,dtInTime,
    strYGRole,strYGDuty,strYGState,strYGNation,strYGDepart);

    //verify return;
    CPPUNIT_ASSERT(bVal==TRUE);

    //verify data table
    CString strSql;
    strSql.Format("SELECT * from t_yg where ygbh=(select max(ygbh) from t_yg)");

    CADORecordset * pRs=new CADORecordset(theApp.pHr_AdoDb);
    if (pRs->Open((LPCTSTR)strSql)) {
        CString dbName;
        COleDateTime dbBirth;
        CString dbState;

        pRs->GetFieldValue("YGXM",dbName);
        CPPUNIT_ASSERT_EQUAL(CString(strYGName),dbName);

        pRs->GetFieldValue("YGSHR",dbBirth);
        CPPUNIT_ASSERT_EQUAL(dtBirthDay,dbBirth);

        pRs->GetFieldValue("ZYZT",dbState);
        CPPUNIT_ASSERT_EQUAL(strYGState,dbState);
    }

    pRs->Close();
    delete pRs;
}
```

IsExist

```
void CMClerkTestCase::IsExist()
{
    BOOL bVal=clerkFixture.IsExist(strYGPriCode);
    CPPUNIT_ASSERT(bVal==TRUE);
}
```

SetClerkSex

```
void CMClerkTestCase::SetClerkSex()
{
    int nMaxCode;
    GetMaxClerkCode(nMaxCode);
```

```
    CString strCode;
    strCode.Format("%d",nMaxCode);

    BOOL bVal=clerkFixture.SetClerkSex(strCode,"女");
    CPPUNIT_ASSERT(bVal==TRUE);

    //verify data table
    CString strSql;
    strSql.Format("Select * From T_YG Where YGBH=%s",strCode);

    CADORecordset * pRs=new CADORecordset(theApp.pHr_AdoDb);
    if (pRs->Open((LPCTSTR)strSql)) {
        CString dbSex;

        pRs->GetFieldValue("YGXB",dbSex);
        CPPUNIT_ASSERT_EQUAL(CString("女"),dbSex);

        pRs->Close();
        delete pRs;
    }
}
```

SetClerkState

```
void CMClerkTestCase::SetClerkState()
{
    int nMaxCode;
    GetMaxClerkCode(nMaxCode);
    CString strCode;
    strCode.Format("%d",nMaxCode);

    BOOL bVal=clerkFixture.SetClerkState(strCode,"辞退");
    CPPUNIT_ASSERT(bVal==TRUE);

    //verify data table
    CString strSql;
    strSql.Format("Select * From T_YG Where YGBH=%s",strCode);

    CADORecordset * pRs=new CADORecordset(theApp.pHr_AdoDb);
    if (pRs->Open((LPCTSTR)strSql)) {
        CString dbState;
        pRs->GetFieldValue("ZYZT",dbState);
        CPPUNIT_ASSERT_EQUAL(CString("辞退"),dbState);
```

```
        pRs->Close();
        delete pRs;
    }
}
```

RemoveClerk

```
void CMClerkTestCase::RemoveClerk()
{
    int nMaxCode;
    GetMaxClerkCode(nMaxCode);
    CString strCode;
    strCode.Format("%d",nMaxCode);

    BOOL bVal=clerkFixture.RemoveClerk(strCode,strYGName,strYGPriCode);
    CPPUNIT_ASSERT(bVal==TRUE);

    //verify data table
    CString strSql;
    strSql.Format("Select * From T_YG Where YGBH=%s",strCode);

    CADORecordset * pRs=new CADORecordset(theApp.pHr_AdoDb);
    if (pRs->Open((LPCTSTR)strSql)) {

    BOOL bVal=pRs->IsEOF();
    CPPUNIT_ASSERT(bVal==TRUE);

    pRs->Close();
    delete pRs;
  }
}
```

在 CppUnit 的用例代码中，主要以测试断言的形式来进行测试的输出。CppUnit 应用最多的两个方法是 CPPUNIT_ASSERT 和 CPPUNIT_ASSERT_EQUAL，通过预期结果与实际运行结果的比较获得测试结果信息。只要稍加留意，就会发现其间也闪现着测试用例的设计思想，这与之前论述的测试用例部分的内容是吻合的。所以从来都不要忽视对理论基础的学习，扎实的理论基础有助于你的快速提高，举一反三是靠知识实现的。

5. 单元测试最佳实践

1）谁做单元测试更合适

通常单元测试是在软件产品的编码阶段进行的，单元测试大部分工作是编写测试代码，这对于测试人员是一个挑战，要求测试人员要有一定的编程技能。

事实上在大部分企业或组织，单元测试的工作主要是以开发人员为主的，这是因为开发人员从事单元测试比测试人员有着得天独厚的优势。这是因为开发人员介入业务需求、系统设计等的工作要远远早于测试人员，开发人员的编程技能因为工作性质的关系要比测试人员强，编写测试代码的效率高。

当然，以上分析只是基于目前整体产业现状所做的阐释，从职业发展的长期性来看，测试人员和开发人员之间的界限会越来越模糊，测试人员掌握必要的测试技能，并能够从事单元测试的编码工作对于测试工作的顺利开展有益无害，相反会更加促进其对软件测试的理解。

2）单元测试的广度和深度

通过前面的内容可以看出，单元测试的工作的复杂度和工作量远超过单元的构建开发过程，实施全面的单元测试对于以逐利为核心的企业来说是很难接受的。虽然目前可以找到很多号称先进的软件单元测试工具或者说框架（FrameWork），但是总体来说，相对于单元测试的复杂性而言，这些工具本身仍然具备充足的提升空间，单元测试的效率和效果有赖于单元测试的执行者。面对 ROI，单元测试实践几乎沦为纸上谈兵。那么如何破局呢？裁剪成了很多企业不约而同的选择，在有限的测试资源投入下，对单元测试不能求全责备，建议企业如果实施单元测试，则至少应做到以下几点：

① 结构优化优先。可测性现在被提到的频率越来越高，可测性也是衡量架构设计是否合理和代码质量的指标之一。能够以最简单的方式测试模块无疑是节约测试投入的最有效方式。

② 核心业务优先。抽取核心业务进行单元测试。

③ 公用模块优先。每一次迭代、每一次版本发布中，相对稳定的部分或者被调用次数最多的部分。

④ 分层测试，中间层优先。多层架构，中间层测试优先。中间层承上启下，一旦存在缺陷造成系统瘫痪的概率最大，在风险评级中也属于风险最高的一类。

6．单元测试的好处及其误解

1）单元测试的好处

（1）尽早测试。

单元测试是在软件编码阶段就开始的测试工作，符合软件测试尽早进行的原则。

模块中的每一个功能都有对应的测试代码来验证其正确性，只要保持需求的一致性，内部代码的处理逻辑如何变更，都不会影响测试代码的运行，即可以做到一套测试用例的多次复用，对于后续的测试可以节约时间。

（2）单元测试是一种测试角度的设计行为，有助于提高代码质量。

单元测试需要编写测试代码以验证产品代码的正确性，编写测试代码的过程是一种从调用者观察、思考产品架构的过程。在 TDD 开发模型中，先写测试用例，再编程序成为一条最主要的原则，这种新的开发模式将改变开发人员的固有编程习惯，而努力把程序设计成易于调用和可测试的。

（3）单元测试也可以理解为是一种编写文档的行为。

单元测试是一种无价的文档，它是展示函数或类如何使用的最佳文档。这份文档相

比于传统的文档有着不可比拟的优势，因为它是可编译、可运行的，是一种会说话的文档。

2）有关单元测试的几种误解

（1）单元测试就是白盒测试。

单元测试的主要方法属于白盒测试的范畴，但是单元测试不能等同于白盒测试，事实上有时候单元测试也会辅助使用一些黑盒测试的技巧。

（2）进行单元测试是浪费时间。

单元测试符合尽早测试的软件测试原则，也是最有效的软件测试手段之一，有统计显示大约 70％～80％左右的缺陷是在单元测试期间发现的。

（3）测试代码并不是我的工作。

这个问题本质上仍然是单元测试谁来做的问题，关于这个问题我们在前面章节已经给出答案。从对产品质量负责的角度，我们认为每个人都有责任。

10.3.2 集成测试

1. 概述

集成测试（也叫组装测试、联合测试）是单元测试的逻辑扩展。集成测试是在单元测试的基础上，测试在将所有的软件单元按照概要设计规格说明的要求组装成模块、子系统或系统的过程中，各部分工作是否达到或实现相应技术指标及要求的活动。也就是说，在集成测试之前，单元测试应该已经完成，集成测试中所使用的对象应该是单元测试已经通过的单元。这一点很重要，因为如果不经过单元测试，那么集成测试的效果将会受到很大影响，并且会大幅增加软件单元代码纠错的代价。

2. 测试内容和方法

集成测试是各个已通过单元测试的模块的组装和验证过程，基于此，集成测试的测试关注点主要包含以下几点：

① 在把各个模块组装起来的时候，各模块间的接口是否按照设计规格要求运行。

② 模块集成后，各个模块间相互作用相互影响，一个模块的功能是否会对另一个模块的功能产生不利的影响是集成测试第二个关注点。

③ 子功能组装后可以把它视为一个更大的子模块或者说系统配置项，这个系统配置项是否达到了其规格说明要求是集成测试的第三个内容。

④ 全局数据结构是否有问题。

⑤ 单个模块的误差累积起来，是否会产生放大效应，以至于导致整个系统不可用。

⑥ 单个模块的错误是否会导致数据库错误（数据丢失、数据库死锁等）。

3. 集成方案介绍

选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号的次序和测试的次序，以及生成测试用例的费用和调试的费用。集成测试的实施方案有很多种，如自底向上集成测试、自顶向下集成测试、Big-Bang 集成测试、三明治集成测试、核心集成测试等。

1）自底向上的集成

自底向上的集成(bottom-up integration)方式是最常使用的方法。其他集成方法都或多或少地继承、吸收了这种集成方式的思想。

自底向上集成方式从程序模块结构中最底层的模块开始组装和测试。因为模块是自底向上进行组装的，对于一个给定层次的模块，它的子模块(包括子模块的所有下属模块)事前已经完成组装并经过测试，所以不再需要编制桩模块(有关桩模块和驱动模块，我们在讲述单元测试的内容时已经做出过解释)。

自底向上集成测试的步骤大致如下：

① 按照概要设计规格说明，明确有哪些被测模块。在熟悉被测模块性质的基础上对被测模块进行分层，在同一层次上的测试可以并行进行，然后排出测试活动的先后关系，制定测试进度计划。

② 在步骤一的基础上，按时间线序关系，将软件单元集成为更大一级的模块，并记录和报告在集成过程中出现的问题。

③ 将各软件模块集成为子系统(或分系统)，检测各子系统是否能正常工作。

④ 将各子系统集成为目标系统，测试目标系统的整体运行情况。

【点评】

优点：管理方便、测试人员能较好地锁定软件故障所在位置，符合测试尽早介入的原则。

缺点：可能要求编写一定数量的驱动模块。

2）自顶向下集成

将模块按系统程序结构，沿控制层次自顶向下进行集成。这种增殖方式在测试过程中较早地验证了主要的控制和判断点。在一个功能划分合理的程序结构中，判断常出现在较高的层次，较早就能遇到。如果主要控制有问题，尽早发现它能够减少以后的返工。

【点评】

优点：能够较早地发现存在于程序主模块中的问题。

缺点：需要建立较多的桩模块，工作量巨大。另外，较为复杂的算法和真正输入/输出的模块一般在底层，它们是最容易出问题的模块，如果到组装和测试的后期才集成和测试这些模块，一旦发现问题，问题修改的成本高昂。

3）Big-Bang 集成测试

非渐增式集成测试的一种策略，测试的时候把所有系统的组件一次性组合成系统进行测试。

【点评】

优点：可以直接针对目标系统进行测试，测试完成后目标系统基本达到可用状态。

缺点：是一种理想状态，事实上多数软件产品都是不断迭代的产物。等待万事具备再行测试往往导致时间的大量浪费，且违反了测试尽早介入的原则，导致问题发现过于推后，问题修改成本高昂。

4）三明治集成测试(sandwich integration)

三明治是一种形象的称呼，是形容自两头向中间组装的系统集成方式。

【点评】

优点：兼顾了自顶向下和自底向上两种集成测试的优点。

缺点：问题可能会集中于中间模块，导致系统上下脱节。

5）核心集成测试

核心系统集成测试法的思想是先对核心软件部件进行集成测试，在测试通过的基础上再按各外围软件部件的重要程度逐个集成到核心系统中。每次加入一个外围软件部件都产生一个产品基线，直至最后形成稳定的软件产品。核心系统先行集成测试法对应的集成过程是一个逐渐趋于闭合的螺旋形曲线，代表产品逐步定型的过程。

【点评】

优点：核心集成测试能够优先保证目标系统的核心部分，一定程度上规避了 Big-Bang 集成方式的缺点。

缺点：对系统架构要求较高，比如目标系统应该能够明确区分核心软件部件和外围软件部件，核心软件部件应具有较高的耦合度，外围软件部件内部也应具有较高的耦合度，但各外围软件部件之间应具有较低的耦合度。

系统集成方案有很多种，但是没有哪一种方案是完美的，有鉴于此，在实践中系统集成时往往是多种集成方式组合起来使用，这里介绍两种以供学习和参考：

- 衍变的自顶向下的增殖测试：它的基本思想是强化对输入/输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增殖测试。
- 自底向上-自顶向下的增殖测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

4. 应用案例剖析

集成测试(本例是面向对象的集成测试)一般是这样的：

① 确定集成方案，为参与集成的单元(本例中的最小单元视为类)确定一种合适的集成方案。

② 版本确认，集成簇，植入测试代码(比如断言)。

③ 构造测试数据，比较实际运行结果和预期结果，记录差异。

④ 修改可能的缺陷，再次进行集成测试，直至达到集成准出标准。

(1) 集成测试方案选择。

集成测试的策略很大程度上取决于系统的集成方案，因此在进行集成测试前首要的任务是为系统选择一套合适的集成方案。

集成方案主要取决于开发进度和系统设计，从本项目的开发计划中(如图 10-26 所示)，我们可以发现在 2008 年 12 月 18 日前，系统完成的开发任务有：

9		⊟ **系统开发**	**15 工作日?**	**2008年12月8日**	**2008年12月25日**	
10		⊟ **员工信息管理**	**12 工作日?**	**2008年12月11日**	**2008年12月25日**	
11	▦	基本信息管理	2 工作日?	2008年12月11日	2008年12月12日	
12	▦	部门调动	2 工作日?	2008年12月11日	2008年12月12日	
13	▦	员工培训记录	1 工作日?	2008年12月17日	2008年12月17日	
14	▦	员工信息变更历史记录	2 工作日?	2008年12月16日	2008年12月17日	
15	▦	合同管理	4 工作日?	2008年12月19日	2008年12月23日	
16	▦	工作履历管理	4 工作日?	2008年12月22日	2008年12月25日	
17		⊟ **培训信息管理**	**4 工作日?**	**2008年12月15日**	**2008年12月18日**	
18		培训信息管理	2 工作日?	2008年12月17日	2008年12月18日	19
19	▦	培训机构管理	2 工作日?	2008年12月15日	2008年12月16日	
20		⊟ **系统管理**	**4 工作日?**	**2008年12月8日**	**2008年12月11日**	
21	▦	组织结构管理	4 工作日?	2008年12月8日	2008年12月11日	
22	▦	系统用户管理	4 工作日?	2008年12月8日	2008年12月11日	
23	▦	岗位及岗位授权管理	4 工作日?	2008年12月8日	2008年12月11日	

图 10-26　系统开发计划

- 员工信息管理：添加、修改(离职管理、退休管理以及其他信息变更等)和删除。
- 培训信息管理：培训信息添加、修改和删除。
- 培训机构管理：培训机构添加、修改和删除。

在 2008 年 12 月 25 日前，系统完成的开发任务有：

- 员工合同管理：合同添加、修改和删除。
- 员工履历管理：工作履历添加、修改和删除。

系统提供的设计方案是这样的(见图 10-27)：配置项 P01_01_JBXX 由配置项 P01_02_HTGL、P01_03_PXJL 和 P01_04_GZLL 组成，而配置项 P01_03_PXJL 又由配置项 P02_PXXX 组成，P02_01_PXX 和 P02_01_PXJG 集成组成配置项 P02_PXXX。

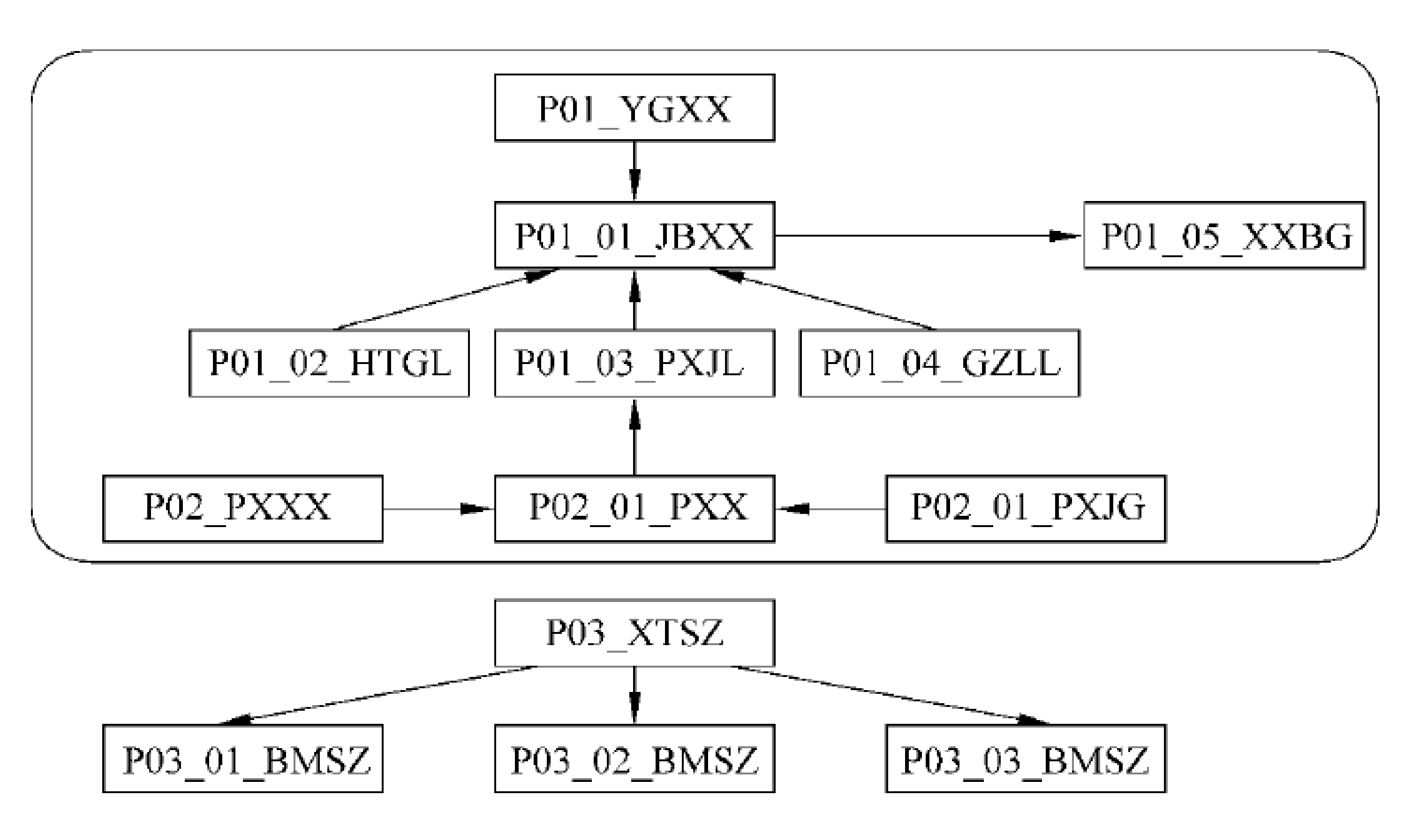


图 10-27　系统层次结构图

根据这种配置项间关系的分析以及开发进度，可以判定该系统适用于自顶向下的集成方案，简单描绘出系统的整体集成模型，如图 10-28 所示。

假如，我们是 12 月 25 日后介入集成测试，则参与集成的模块有 5 个。通过分析类图手册或者源程序可以找出所有这些参与集成的对象类。根据类之间的聚合关系，绘出类的集成关系如图 10-29 和图 10-30 所示，其中图 10-30 对应集成模块培训信息管理和培训机构管理，图 10-30 对应集成模块员工信息管理、培训记录查询、员工合同管理和员工履历管理。

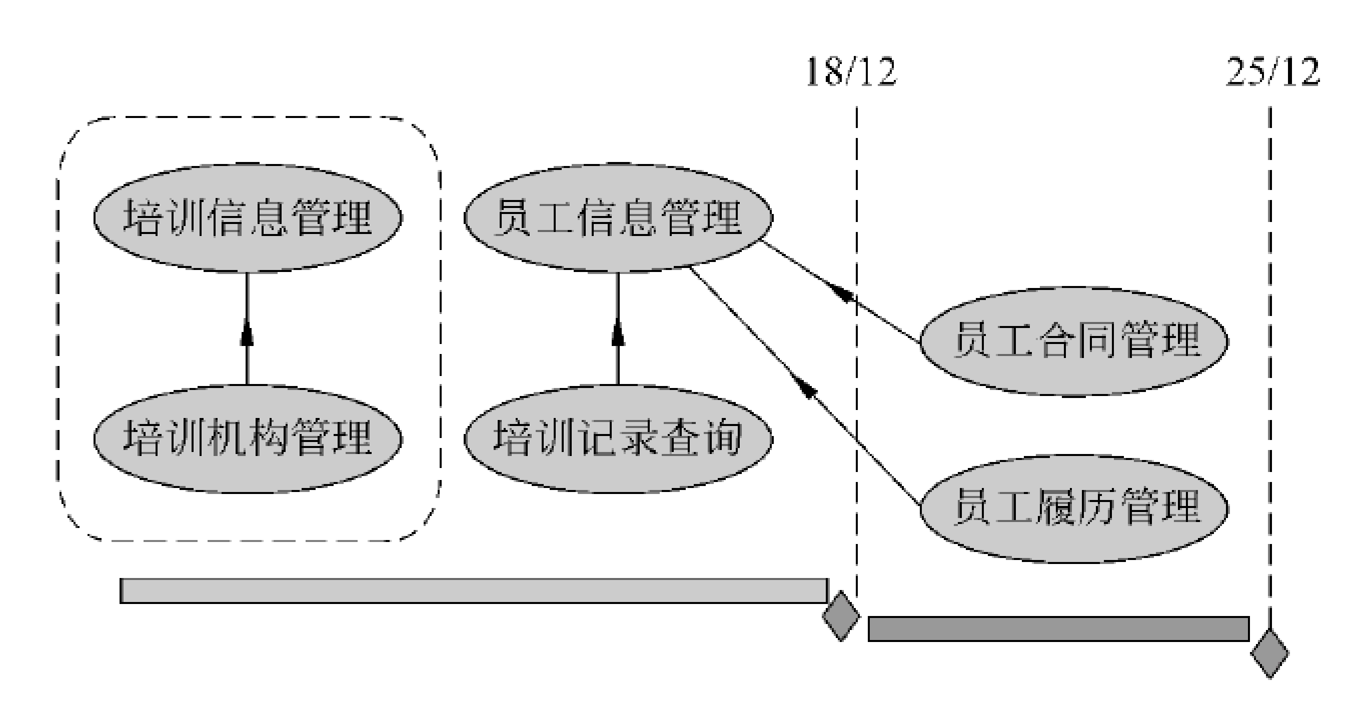


图 10-28　系统集成示意

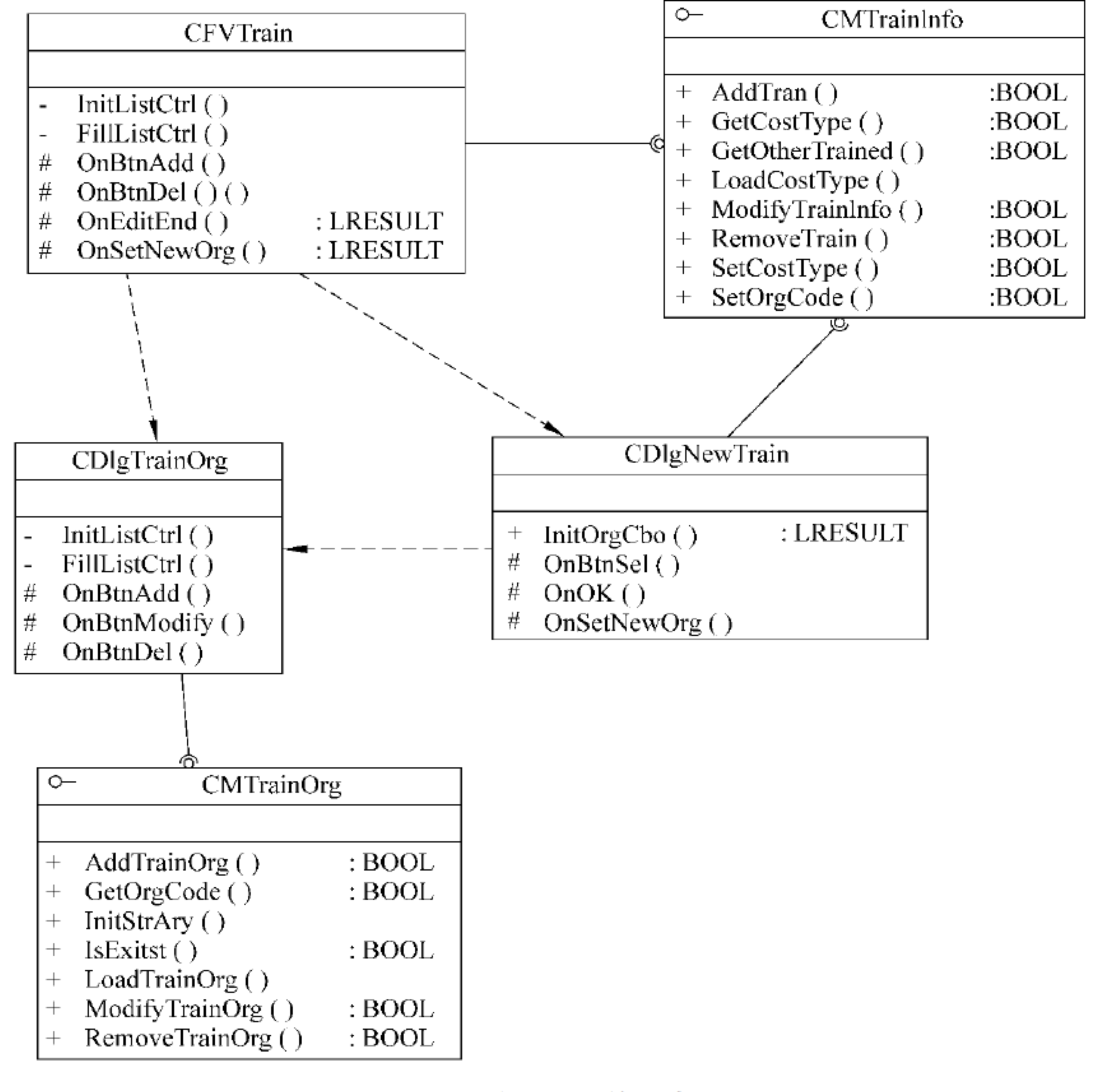


图 10-29　培训信息管理类关系

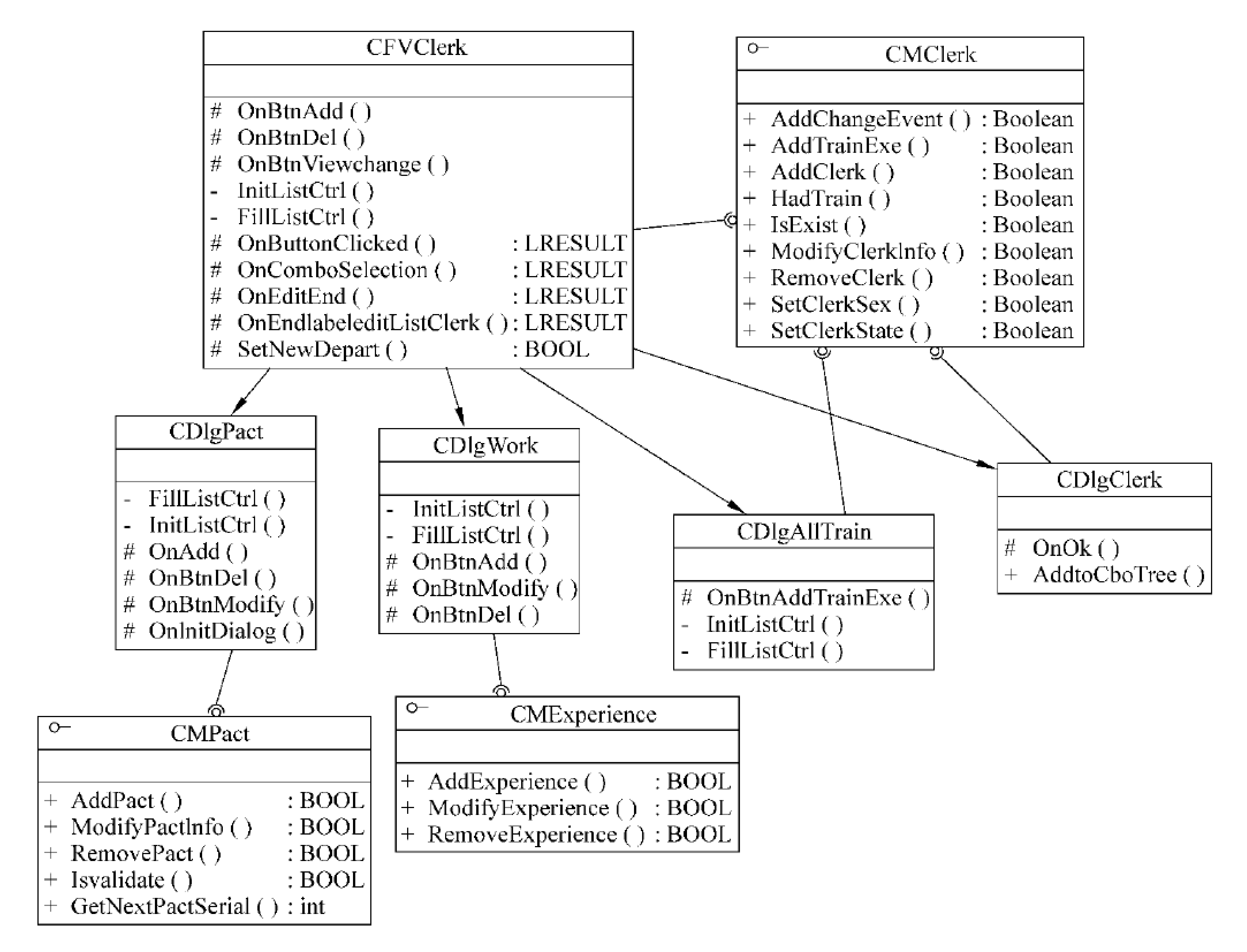


图 10-30　员工信息管理类关系

所有与培训相关的对象类相互间通过方法的调用实现消息的转移，有一定的相依性，这些相互关联的类聚集在一起，构成一个"簇"。这个簇有个顶级类 CFVTrain，在 HRMIS 的迭代开发过程中，通过 CFVTrain 完成对下一层增量对象类 CDlgNewTrain、CMTrainInfo、CDlgTrainorg 的调用，CDlgTrainorg 负责调用再下一层的类 CMTrainOrg，这是一种典型的广度优先的自顶向下的集成模式。再如图 10-30 这个集成簇，顶层控制类是 CFVClerk，CFVClerk 负责调用 CDlgAllTrain、CDlgPact、CDlgWork 和 CMClerk，第二层的类 CDlgPact 和 CDlgWork 分别调用下一层的类 CMPact 和 CMExperience。

(2) 集成测试设计和执行。

集成测试的重点是验证参与集成的类间消息传递的准确性，下面来看图 10-30 所示的这个集成簇的集成测试用例的设计。

首先来分析这个簇所包含的各个类之间的协作关系，分解如下：

	CFVClerk	CMClerk	CDlgClerk	CDlgWork	CMExperience	CDlgPact	CMPact	CDlgAll-Train
1	OnEditEnd	ModifyClerkInfo						
		SetClerkSex						
		SetClerkState						
	OnBtnDel	RemoveClerk						
	OnBtnAdd		DoModal					
	OnButtonClicked			DoModal		DoModal		DoModal
2		AddClerk	OnOK					
		AddTrainExe						OnBtnAdd TrainExe
				OnBtnAdd	AddExpe-rience			
				OnBtnModify	ModifyExp-erience			
				OnBtnDel	RemoveExp-erience			
						OnAdd	AddPact	
						OnBtn-Modify	ModiyfPac-tInfo	
						OnBtnDel	Remove-Pact	

这个表称为协作关系矩阵，使用协作关系矩阵可以保证集成测试的全面性，防止漏测。从这个协作关系矩阵图中，可以很清晰地看到在 CFVClerk 和 CMClerk 之间存在着这样一些协作关系（矩阵图左上角）。

CFVClerk	CMClerk
OnEditEnd	ModifyClerkInfo
	SetClerkSex
	SetClerkState
OnBtnDel	RemoveClerk

集成测试的测试用例可以从外部行为入手进行测试，例如信息系统可以通过对结果数据的验证来确认接口的可用性和准确性。本例在设计测试用例时可以使用测试断言的形式对接口进行确认。所谓测试断言可以是以下形式的测试代码：

- 接收数据的打印输出；
- 接收数据与预期数据的比较，相同时给出成功提示信息，不相同时给出警告信息。

在本例中，要对类接口 ModifyClerkInfo 进行测试，设计测试用例如下：

【用例 10-7】

用例名称	测试修改员工信息	用例标识	P200901UT-YGGL-YGLR-07	
测试追踪	—			
用例说明	测试修改指定员工信息，CMClerk 能否准确接收到新的信息：可供修改的信息有员工名称、姓名、性别、政治面貌等信息			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在员工信息列表中选择一个员工并修改其各项信息（包括姓名、性别、年龄、出生日期、身份证号等）	—	—	
2	确认修改	—	—	
3	查看信息修改是否正确	—	信息已成功修改并更新至数据库	
前提和约束		• 查询系统中已存在的员工且员工记录不少于 1 条 • 所输入的新的身份证号不能重复		
过程终止条件		无		
结果评价标准		信息已成功修改并更新至数据库		
设计人员		—	设计日期	—

为了方便对测试信息进行确认，在接口函数 ModifyClerkInfo 中增加一个简单的消息类型的测试断言（如图 10-31 所示）。当然，过多的这种测试断言会大大地阻碍集成测试的进程，所以一般可以采取打印输出的方案，即定义一个统一的信息输出管道，将需要验证的数据信息统一输出到指定文件中，这样可以在执行完所有测试用例后再统一对数据进行分析。

理论上，在函数执行前进行信息确认基本上就可以达到对接口的确认目的，因为在介绍单元测试时，已经对 CMClerk 这个类进行了严格的单元测试，对于函数本身处理逻辑的正确性应该可以得到确认了。

```
CADORecordset* pRs = new CADORecordset(theApp.pHr_AdoDb);
pRs->Open("T_YG", CADORecordset::openTable);
pRs->SetFilter(strSQL);
pRs->Edit();

//Test Assert
AfxMessageBox(ygModified);

switch(mPointer) {
// 1/3/4 存到另外的表中
case 2:/*部门编号*/
    pRs->SetFieldValue("BMBH",ygModified);
    break;
```

图 10-31　消息代码

10.3.3　系统测试

1. 概述

系统测试就是把目标软件植入其实际或者模拟运行环境中,将整个运行体系作为一个系统进行仿真测试的过程。系统测试是整个软件测试过程的一个重要阶段,它开始于单元测试、集成测试之后,是软件交付最终用户前,软件开发组织进行的最后一道产品质量保障手段。

2. 系统测试的内容

系统测试的一个很大作用是随着测试进程逐渐揭示系统的外在质量属性,逐渐建立起产品组对自身产品的信心。在系统测试期间,不仅仅要对功能进行验证,另外有关软件产品的非功能测试要求和性能、负载、压力等(系统测试关注的质量特性如图 10-32 所示)都要在这个阶段进行测试。

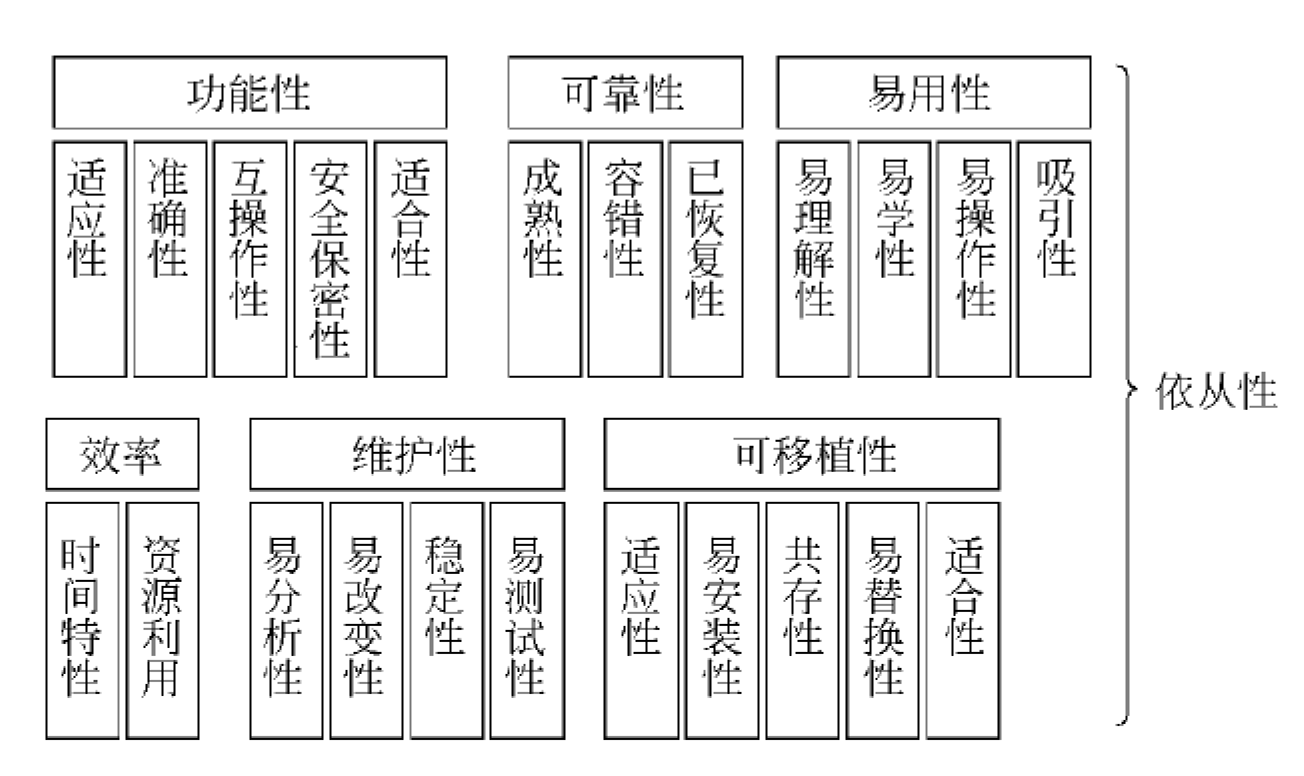


图 10-32　系统测试关注的质量特性

系统测试具体包括以下内容:

1) 功能测试

功能测试是在模拟环境(软件测试要求独立搭建测试环境,不能与开发共用一个环境)下,运用黑盒测试方法,验证被测软件是否满足需求规格说明书列出的需求。功能测试最基本的方法是基于测试用例进行,即针对系统的功能项预先编制测试步骤,设计测试数据。通过实施预定测试步骤和测试数据,确定软件的特性是否与预期结果一致,再据此判断与客户需求的相符性,确保所有的软件功能需求都能得到满足。

2）非功能测试

非功能测试从效率角度一般关注事务平均响应事件、并发用户数以及服务器资源利用率等性能指标，这些指标的获取一般通过性能测试、负载测试、压力测试等性能测试手段取得。

非功能测试同时也关注系统的稳定性，比如 MTTF（平均无故障时间）、MTTR（平均故障恢复时间）等。

3）文档测试

文档测试的主要依据是企业自己制定的文档规范，另外也可以参考相关文档编制规范（如第 1 章介绍的 GB/T 8567—2006《计算机软件文档编制规范》）。

文档测试的要点可以概括为以下几点：

- 应该与系统提供的功能进行核对，用户文档应包含产品使用所需的全部信息；
- 检查文档前后说明的内容是否自相矛盾；
- 文档中信息表达正确，不含有歧义和错误的表达，同时与实际功能进行比对，确认描述内容是否正确，文档是否易于浏览。

3．系统测试的技术要求

系统测试一个最为典型的特征是基于系统外部行为特征的规格说明（可以理解为产品需求规格说明）的全面测试，不依赖于系统内部结构设计。系统测试阶段一般无须编写更多测试用例，这个阶段的用例大多直接衍生于系统的架构和设计规格、最终用户以及用户故事等，系统的每个功能有可能会衍生一个或者多个测试用例。系统测试阶段也可以引用单元测试、集成测试阶段的用例，但是应该掌握这样一个原则：即应该首先保障系统测试用例执行，而不是重复执行单元、集成测试用例。

有关系统测试的用例设计通常有以下技术要求：

① 系统的每一个功能点应至少被一个正常测试用例和一个被认可的异常测试用例所覆盖。

② 测试用例的输入应至少包括有效、无效等价类。

③ 多使用边界值进行测试，边界值可以使用恰好等于、稍微大于、稍微小于这样一些经验证明经常出现问题的用例数据。

系统测试阶段发现的缺陷处理起来要求更加谨慎，在修复之前应该做好充分的影响性分析。很多时候，如果当前业务可以暂时容忍这些缺陷的存在，则可做如下处理：暂时不修改缺陷，但是须记录并报告这些缺陷，并在产品说明书中作为限制性条款指明软件的局限性。

4．应用案例剖析

1）功能确认测试

根据 HRMIS 项目的测试计划，要对以下业务功能进行测试（如图 10-33 所示）。

下面以本案例中的“员工信息管理→新增员工信息”这个功能的测试为例，看一下前面介绍的黑盒测试方法是如何应用到测试用例的设计开发过程的。

从 HRMIS 的系统需求说明书中可以看到一条完整的员工信息包括员工的姓名、性

	任务名称	工期	开始时间	完成时间	
22	⊟ **系统测试**	**18 工作日?**	**2008年12月29日**	**2009年1月19日**	
23	⊟ **功能测试**	**3 工作日?**	**2008年12月29日**	**2008年12月31日**	
24	⊟ **员工信息管理**	**2 工作日?**	**2008年12月29日**	**2008年12月30日**	
25	新增员工信息、修改员工信息和删除员工信息	1 工作日?	2008年12月29日	2008年12月29日	
26	部门调动	1 工作日?	2008年12月29日	2008年12月29日	
27	⊟ **合同管理**	**1 工作日?**	**2008年12月29日**	**2008年12月29日**	
28	合同签订	1 工作日?	2008年12月29日	2008年12月29日	
29	合同修改	1 工作日?	2008年12月29日	2008年12月29日	
30	合同删除	1 工作日?	2008年12月29日	2008年12月29日	
31	⊟ **工作履历**	**1 工作日?**	**2008年12月30日**	**2008年12月30日**	
32	添加工作履历	1 工作日?	2008年12月30日	2008年12月30日	
33	修改工作履历	1 工作日?	2008年12月30日	2008年12月30日	
34	删除工作履历	1 工作日?	2008年12月30日	2008年12月30日	
35	参加培训记录	1 工作日?	2008年12月30日	2008年12月30日	
36	信息变更历史记录	1 工作日?	2008年12月30日	2008年12月30日	
37	⊟ **培训信息管理**	**1 工作日?**	**2008年12月31日**	**2008年12月31日**	
38	培训信息添加、修改和删除	1 工作日?	2008年12月31日	2008年12月31日	
39	培训机构添加、修改和删除	1 工作日?	2008年12月31日	2008年12月31日	
40	培训信息的培训机构变更	1 工作日?	2008年12月31日	2008年12月31日	
41	⊟ **系统管理功能测试**	**1 工作日?**	**2009年1月1日**	**2009年1月1日**	
42	系统用户添加、修改和删除	1 工作日?	2009年1月1日	2009年1月1日	
43	系统用户权限设置	1 工作日?	2009年1月1日	2009年1月1日	
44	问题修改与跟踪	2 工作日?	2009年1月1日	2009年1月2日	
45	首次回归测试	5 工作日?	2009年1月3日	2009年1月8日	44

图 10-33 HRMIS 系统测试计划

别、民族、出生日期、婚姻状况、政治面貌、文化程度、家庭住址、身份证号、毕业院校、籍贯、户口所在地、到岗日期、工作岗位、职务、职责、部门、工作证号、手机号码、办公电话、职业状态、紧急联系人、联系方式、备注信息、照片等共 25 项，其中姓名、性别、身份证号、职业状态是必填项。员工的身份证号是唯一的，不能重复。员工信息成功添加后，系统自动生成一个唯一的序号。

测试用例的设计思路：

（1）确定测试要点。对被测功能输入数据的规格和约束，处理机制，操作顺序，输出结果进行分析，在分析时需要考虑数据长度、类型、大小、数据项之间的制约关系等约束，在此分析的基础上，确定出被测功能的测试要点。

（2）测试用例设计。针对测试要点进行测试用例设计，加入测试时使用的测试数据，描述出预计的测试结果，形成测试用例。

对于增加界面中的每一个输入项，可以确定以下测试要点：数据长度验证、数据类型验证，可以采用等价类划分、边界值分析来进行测试用例设计。

例如对于“姓名”，其数据要求为：只允许输入字母、汉字，数据长度限制为[1,12]个半角字符。从中可以看出，包含了数据类型验证，数据长度验证。

从数据类型的角度出发，根据等价类划分法，可以设定一个有效等价：输入字母或汉字，若干个无效等价类：输入数字、输入标点、输入特殊字符等。据此我们至少可以得到用例 10-8～10-10 3 个测试用例。

【用例 10-8】

用例名称	增加员工记录	用例标识	P200901UT-YGGL-YGLR-08
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	增加一条记录，“姓名”输入“张三”，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据构成划分的有效等价类		

续表

用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击"新增"按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入"张三"，其他数据栏输入相应信息	—	—	
4	单击"保存"按钮	员工信息成功保存	可以查询到新录入的员工信息	
前提和约束		所输入的员工身份证号不能与已有员工信息重复		
过程终止条件		无		
结果评价标准		查询到的员工信息与录入信息保持一致		
设计人员		—	设计日期	—

【用例 10-9】

用例名称	增加员工记录		用例标识	P200901UT-YGGL-YGLR-09
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	增加一条记录，"姓名"输入"abc"，其他输入项分别输入符合要求的数据后，单击"保存"按钮。用例设计方法：以数据构成划分的有效等价类			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	

续表

序号	输入及操作说明	期望的测试结果	评价标准	备 注
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“abc”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息成功保存	可以查询到新录入的员工信息	
前提和约束		所输入的员工身份证号不能与已有员工信息重复		
过程终止条件		无		
结果评价标准		查询到的员工信息与录入信息保持一致		
设计人员		—	设计日期	—

【用例 10-10】

用例名称	增加员工记录	用例标识	P200901UT-YGGL-YGLR-10
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	增加一条记录，“姓名”输入“!? ＜＞”，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据构成划分的无效等价类		
用例的初始化	硬件配置	无特殊要求	
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库	
	测试配置	无特殊要求	
	参数设置	部门组织结构已初始化	

操作过程

序号	输入及操作说明	期望的测试结果	评价标准	备 注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“!? ＜＞”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息保存失败	系统给出错误提示，查询不到新录入的员工信息	

续表

前提和约束	所输入的员工身份证号不能与已有员工信息重复		
过程终止条件	无		
结果评价标准	查询不到新录入的员工信息且对原有数据信息无影响		
设计人员	—	设计日期	—

另外，还可以从数据长度的角度出发，根据边界值分析法，数据长度的边界是1个半角字符、12个半角字符，根据等价类划分法，可以设定一个有效等价类：1个半角字符≤姓名长度≤12个半角字符；两个无效等价类：姓名长度<1个半角字符，姓名长度>12个半角字符。据此可以得到用例表（见表10-15）：

表10-15 用例表

序号	输入及操作说明	期望的测试结果	备注（所采用的方法）
1	增加一条记录，“姓名”为空，其他输入项分别输入符合要求的数据后，单击“保存”按钮	系统给出提示，数据保存失败	边界值、无效等价类，输入0个字符
2	增加一条记录，“姓名”输入“a”，其他输入项分别输入符合要求的数据后，单击“保存”按钮	记录增加成功，通过查询可以查到该记录	边界值，输入一个半角字符
3	增加一条记录，“姓名”输入“张三 ab”，其他输入项分别输入符合要求的数据后，单击“保存”按钮	记录增加成功，通过查询可以查到该记录	有效等价类，输入6个半角字符
4	增加一条记录，“姓名”输入“张三李四王 a”，其他输入项分别输入符合要求的数据后，单击“保存”按钮	记录增加成功，通过查询可以查到该记录	边界值，输入11个半角字符
5	增加一条记录，“姓名”输入“张三李四王五”，其他输入项分别输入符合要求的数据后，单击“保存”按钮	记录增加成功，通过查询可以查到该记录	边界值，有效等价类12个半角字符
6	增加一条记录，“姓名”输入“张三李四王五 a”，其他输入项分别输入符合要求的数据后，单击“保存”按钮	系统给出提示，数据保存失败	边界值，输入13个半角字符

作为示例，其中1和5的完整测试用例描述参见用例10-11和用例10-12。

【用例10-11】

用例名称	增加员工记录	用例标识	P200901-YGGL-YGLR-11
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	增加一条记录，“姓名”为空，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据长度划分的无效等价类		

续表

用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行,正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	姓名栏为空,其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息保存失败	系统给出出错提示,查询不到新录入的员工信息	
前提和约束		所输入的员工身份证号不能与已有员工信息重复		
过程终止条件		无		
结果评价标准		查询不到录入的员工信息且对原有数据无影响		
设计人员		—	设计日期	—

【用例 10-12】

用例名称	增加员工记录		用例标识	P200901-YGGL-YGLR-12
测试追踪	业务需求说明书:人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中,TR01			
用例说明	增加一条记录,“姓名”输入“张三李四王五”,其他输入项分别输入符合要求的数据后,单击“保存”按钮 用例设计方法:以数据长度划分的有效等价类			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行,正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				

续表

序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“张三李四王五”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息成功保存	可以查询到新录入的员工信息	
前提和约束		所输入的员工身份证号不能与已有员工信息重复		
过程终止条件		无		
结果评价标准		查询到的员工信息与录入信息保持一致		
设计人员		—	设计日期	—

其他的输入项可以采取类似“姓名”的分析方法，设计出相应的测试用例。当然，在实际测试过程中，如果这 25 个输入项每一个都进行类似的设计，测试用例数是非常庞大的，时间上可能不允许。通常，在这种情况下，可以根据经验，采取错误推测法，选取这 25 个输入项中的一些容易出错的或者比较关键的输入项进行验证。

可以分析出对于 HRMIS 的员工新增功能来讲，还存在以下测试要点：必填项验证、信息重复验证、生成序号的正确性验证、取消功能验证。下面以必填项验证为例，介绍一下判定表的应用。用例设计的过程如下：

① 确定规则的个数：这里有四个必填项，但只涉及两个条件，即姓名、身份证号，各有两个取值，故应有 $2^2=4$ 种规则。性别、职业状态有默认值，只有 1 个取值。

② 列出所有的条件桩和动作桩。

条件：
- 是否输入姓名？
- 输入性别
- 是否输入身份证？
- 输入职业状态

动作：
- 保存成功
- 系统提示“请输入员工姓名”
- 系统提示“请输入身份证号”

③ 填入条件项、动作项，得到初始判定表（表 10-16）。

表 10-16　初始判定表

		1	2	3	4
条件	输入姓名	Y	N	Y	N
	输入性别	Y	Y	Y	Y
	输入身份证	Y	Y	N	N
	输入职业状态	Y	Y	Y	Y
动作	保存成功	X			
	系统提示“请输入员工姓名”		X		X
	系统提示“请输入身份证号”			X	

④ 合并相似规则,得到优化后的判定表(见表 10-17)。

表 10-17　优化后的判定表

		1	2	3
条件	输入姓名	Y	N	Y
	输入性别	Y	Y	Y
	输入身份证	Y	—	N
	输入职业状态	Y	Y	Y
动作	保存成功	X		
	系统提示“请输入员工姓名”		X	
	系统提示“请输入身份证号”			X

根据判定表,知道这个测试可以使用 3 个测试用例来进行测试。

序号	输入及操作说明	期望的测试结果	备注(所采用的方法)
1	增加一条记录,“姓名”输入“张三”,“性别”选择“男”,“身份证号”输入“140103197903295732”,“职业状态”选择“实习”,其他输入项为空,单击“保存”按钮	记录增加成功,通过查询可以查到该记录	这里其他输入项为空,这样可以同时测试其他输入项是否也存在必填项的约束,这也是一种经验,用到了错误推测法
2	增加一条记录,“姓名”为空,“性别”选择“男”,“身份证号”输入“140103197903295732”,“职业状态”选择“实习”,其他输入项为空,单击“保存”按钮	系统提示“请输入员工姓名”,数据保存失败。	
3	增加一条记录,“姓名”输入“张三”,“性别”选择“男”,“身份证号”为空,“职业状态”选择“实习”,其他输入项为空,单击“保存”按钮	系统提示“请输入身份证号”,数据保存失败	

其中，用例 10-13 的完整用例描述如下：

【用例 10-13】

<table>
<tr><td>用例名称</td><td colspan="2">增加员工记录</td><td>用例标识</td><td>P200901-YGGL-YGLR-13</td></tr>
<tr><td>测试追踪</td><td colspan="4">业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01</td></tr>
<tr><td>用例说明</td><td colspan="4">增加一条记录，“姓名”输入“张三”，“姓别”选择“男”，“身份证号”为空，“职业状态”选择“实习”，其他输入项为空，单击“保存”按钮
用例设计方法：以数据长度划分的无效等价类</td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="3">无特殊要求</td></tr>
<tr><td>软件配置</td><td colspan="3">软件可以正常启动运行，正常连接 MySQL 数据库</td></tr>
<tr><td>测试配置</td><td colspan="3">无特殊要求</td></tr>
<tr><td>参数设置</td><td colspan="3">部门组织结构已初始化</td></tr>
<tr><td colspan="5">操作过程</td></tr>
<tr><td>序号</td><td>输入及操作说明</td><td>期望的测试结果</td><td>评价标准</td><td>备　注</td></tr>
<tr><td>1</td><td>打开员工管理界面</td><td>显示员工信息列表</td><td>员工信息列表中显示已存在的员工信息</td><td></td></tr>
<tr><td>2</td><td>单击“新增”按钮</td><td>打开员工信息输入窗口</td><td>显示员工信息输入窗口</td><td></td></tr>
<tr><td>3</td><td>在姓名栏输入“张三”，性别选择“男”，身份证号为空，职业状态选择“实习”，其他输入项为空</td><td>—</td><td>—</td><td></td></tr>
<tr><td>4</td><td>单击“保存”按钮</td><td>员工信息保存失败提示“输入身份证号”</td><td>查询不到新录入的员工信息</td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">前提和约束</td><td colspan="3">无</td></tr>
<tr><td colspan="2">过程终止条件</td><td colspan="3">无</td></tr>
<tr><td colspan="2">结果评价标准</td><td colspan="3">查询不到录入的员工信息且对其他数据无影响</td></tr>
<tr><td colspan="2">设计人员</td><td>—</td><td>设计日期</td><td>—</td></tr>
</table>

2）性能测试

在 HRMIS 的测试计划中，已经对性能测试需求做了分析，比较明确的有两个单项业务的测试（见图 10-34），一个是系统登录，一个是考勤高峰，性能测试要求是当系统并发 15 个用户时，两项业务的平均响应时间要保持在 10s 以内。信息修改、数据转换和传送这两个测试项实际涵盖了很多业务，经过进一步与客户沟通，结合系统使用情况调查，进一步明确为由员工修改办公电话、培训机构信息修改这两个业务组成的混合场景的平均事务处理时间不高于 15s，两项业务的用户分配比为 1∶1（见图 10-35）。

因为系统的性能测试需求是明确的，因此 HRMIS 的性能测试使用性能符合性能验

性能	系统登录	15用户并发，平均响应时间<10s	
	考勤高峰	15用户并发，平均响应时间<10s	
	信息修改	平均小于15s	确定典型业务
	数据转换和传送	平均小于15s	确定典型业务

图 10-34　HRMIS 系统测试质量需求节选

	⊟ 性能测试	3 工作日?	2009年1月3日	2009年1月6日	
	系统登录独立场景基准测试	1 工作日?	2009年1月3日	2009年1月3日	
	考勤独立场景基准测试	1 工作日?	2009年1月5日	2009年1月5日	70
	员工修改联系电话、修改培训机构信息混合场景测试	1 工作日?	2009年1月6日	2009年1月6日	71

图 10-35　HRMIS 系统测试计划节选

证测试策略。结合测试需求，构造出性能测试场景，其中一个是用户并发登录测试，一个是信息修改的混合交易测试（见表 10-18 和表 10-19）。

5. 系统测试与 SDLC

系统测试与 SDLC（软件开发生命周期）关系密切。系统测试在设计测试用例时，要大量参考需求规格，甚至原始的用户故事等，在这个过程中不仅仅是用例设计的方便，另外还有一个价值，那就是可以及时发现需求中的问题，对需求规格间接进行了测试和文档的优化。但是发现需求中的问题在项目初期最有价值，因此系统测试也应遵循软件测试尽早介入的原则，提早开始系统测试用例的设计编写工作。

另外，系统测试还被某些开发组织应用于开发进度的度量，他们用系统测试用例的通过数量与全部系统测试用例的比值来评估总体软件开发进程，也算是一个有益的尝试。

10.3.4　验收测试

验收测试是一种正式的测试，主要用来判定软件产品是否符合其预定义的验收标准。验收测试与系统测试不同，首先系统测试的执行者总是开发组织，而正式的验收测试一般由最终用户来完成。

验收测试通常以 Alpha 测试和 Beta 测试这两种形式出现，那么什么是 Alpha 测试和 Beta 测试呢？Alpha 测试是最终用户在开发方进行的测试，测试环境有一定限制，一是不可能完全与生产环境一致，另外测试环境受开发组织控制；Beta 测试是完全在客户方进行的测试，测试环境不受开发组织控制。Alpha 测试和 Beta 测试都可以有测试人员参与，以协助最终用户顺利完成测试工作。

在软件开发生命周期的一系列进程中，与客户（软件购买者）达成验收标准是很重要的，一份好的验收计划对于开发组织识别客户的需求是大有裨益的，因此我们在编制和评审验收测试计划时都应尽量争取客户的参与。在 SDLC 过程中开发组织和客户要经常保持有效沟通，并且要确保完成以下事项：

- 识别产品中哪些是临时性的，哪些是要体现在最终产品中的，并依此制定验收标准和验收日程表；
- 规划安排每项验收测试活动由谁采用什么方式执行；
- 给由客户组成的验收小组预留充足的时间，以检审产品；
- 拟制验收计划；

表 10-18　15 用户并发登录系统

场景名称		15 个用户并发登录系统						备　注
数据量		只含有少量初始数据						
总用户数		15						
		用户数	执行步骤	参数化	事务	集合点	检查点	
测试功能	系统登录	15	（1）启动 HRMIS （2）输入用户名和密码 （3）单击“登录”按钮	用户名：{User} 密码：{pword}	T_SysLogin	R_Login	检查 headbar 标题栏，确认登录成功	使用 LoadRunner 的 ODBC 协议录制测试脚本
执行方式	用户加压方式	每秒 5 个用户，直至达到 15 个用户						
	循环执行方式	执行 2 次						
	用户退出方式	自动退出						
执行参数	思考时间	忽略思考时间						
	循环间隔时间	间隔 10s						
	运行日志	默认						
	错误处理	执行期间忽略						
监视指标			指标含义			期望值	监视方法	
	数据库服务器	CPU	%Processor Time：服务器端 CPU 平均使用率			<70%		
		Memory	Available Mbytes：服务器端可利用物理内存			>200MB	LoadRunner 监视器	
		HD	Physical Disk %Disk Time：服务器端磁盘驱动器忙于为读或写入请求提供服务所用的时间的百分比			<80%		
说明		…						

表 10-19　信息修改混合场景测试,20 并发用户

场景名称		员工修改电话(10)、培训机构信息修改(10)混合场景的测试						备　注
数据量		只含有少量初始数据						
总用户数		20						
		用户数	执行步骤	参数化	事务	集合点	检查点	
测试功能	员工修改电话	10	(1) 启动 HRMIS,登录系统 (2) 选择一个员工,修改其办公电话 (3) 保存	用户名:{User} 密码:{pword} 员工编号 员工电话	T_MClerk	无	无(人工验证)	使用 LoadRunner 的 ODBC 协议录制测试脚本
	培训机构信息修改	10	(1) 启动 HRMIS,登录系统 (2) 在培训信息栏单击“新增”按钮,在“新增培训信息”窗口中单击培训机构选择按钮 (3) 在机构信息管理窗口中,勾掉“新增”标识,选择一个培训机构信息 (4) 修改培训机构信息 (5) 保存	用户名:{User} 密码:{pword} 培训机构信息	T_MTrain	无	无(人工验证)	
执行方式	用户加压方式	每秒 5 个用户,直至达到 20 个用户						
	循环执行方式	执行 3 次						
	用户退出方式	自动退出						

续表

执行参数	思考时间		忽略思考时间			
	循环间隔时间		间隔 10s			
	运行日志		默认			
	错误处理		执行期间忽略			
监视指标			指标含义	期望值	监视方法	
	数据库服务器	CPU	%Processor Time:服务器端 CPU 平均使用率	<70%	LoadRunner 监视器	
		Memory	Available Mbytes:服务器端可利用物理内存	>200MB		
		HD	Physical Disk %Disk Time:服务器端磁盘驱动器忙于为读或写入请求提供服务所用的时间的百分比	<80%		
说明			…			

- 在正式递交前，进行正式的验收测试；
- 依据验收测试的结果决策。

前面从纵向角度集中论述了软件测试中的单元测试、集成测试、系统测试和验收测试，为了使大家有一个更为直观的认识，总结概括了有关这几类测试的关键属性做了横向比较(见表 10-20)。

表 10-20　几种类型的测试横向比较

	单元测试	集成测试	系统测试	验收测试
测试用例来源	单元设计规格说明	架构设计规格说明	系统需求规格说明	系统需求规格说明
实施要求	源码清单	部分源代码和大部分接口	对源代码不要求	对源代码不要求
测试环境构建	存在一定复杂性，单元往往不能独立运行，需要构建桩模块和驱动模块支持单元的动态运行	依赖于软件架构和集成顺序。单元或者子系统可以自底向上增量集成以减少对驱动模块和桩模块的需求	这个级别的测试一般不需要桩和驱动的参与，仅仅依赖于测试规则	依赖于测试规则
测试目的	独立模块的内外部行为表现	模块集成后的内外部行为	确认系统功能、性能	从客户角度对系统功能的确认

本章小结

本章围绕着测试用例介绍了软件测试的基本方法：白盒测试和黑盒测试，具体包括：

(1) 白盒测试。

- 代码检查和审查；
- 逻辑覆盖、基本路径测试、数据流测试、程序插装、域测试、程序变异和符号测试等。

(2) 黑盒测试。

- 等价类划分法、边界值分析法、判定表驱动法、因果图；
- 错误推测、正交实验、场景图、功能图。

在介绍以上方法的同时，本章以 HRMIS 的测试为例详细讲解了部分方法是如何在单元测试、集成测试和系统测试中应用的，在进一步理解这些方法的同时也认识了这几类典型测试的实施过程。

CHAPTER

第 11 章 测试实施

运筹帷幄的目的是为了决胜千里。我们在第 10 章对软件测试的设计从方法讲到各类测试的应用方案，可谓详尽，但是测试的成败并不是完全取决于测试设计，测试设计仅仅是为实施一次优秀的测试提供了必要条件，而方案实施的执行情况也是非常重要的，否则再完美的测试方案也只能说是纸上谈兵。我们本章要讲的就是测试的执行，软件生命周期的第四个阶段。相对于其他几个阶段，测试执行也许稍显杂乱，因为测试执行是一个技术加管理的综合过程，为了能够更好地理解测试执行这个过程，我们不妨在进入正式的内容之前，先来试着回答一下以下几个问题：

- 如何保证测试执行人员清楚地知道自己的测试任务和要达到的目标？
- 如何保证测试用例的执行率？
- 如何保证缺陷(bug)描述的准确性，可复现性？
- 如何保证缺陷状态的可跟踪性，不遗漏？
- 如何在验证缺陷或新功能与回归测试之间寻找平衡？
- 如何应对开发计划变更时对测试计划的冲击？
- 如何应对需求发生变更时测试用例的更改甚至废弃？
- 如何保持测试人员与开发人员的有效沟通，合作而不是抵触？

以上这些问题都是测试执行期间的常见问题也是很核心的问题，可以说解决好这几个问题，测试执行基本上就算是合格了。如果我们暂时还找不到合适的答案，那么接下来不妨带着这几个问题去阅读下面的内容，这样也算是在测试执行这个稍显凌乱的领域抓住了一条清晰的线索。

11.1 测试准备

测试执行离不开测试环境，因此我们在测试执行期间的第一步工作就是对测试环境进行确认，也就是准备测试环境所需的硬件设备及软件、搭建测试环境和环境检查。

11.1.1 测试设备检查

测试环境和设备指完成测试工作所需要的设备、设施、软件和网络环境。它应该是独立的,隔离的,不受其他非测试用设备或环境的干扰。

测试环境通常包括软件运行的硬件平台、软件平台、网络平台、其他辅助设备等。硬件平台指软件运行所需要的计算机设备,可以是巨型机、大型机、小型机、服务器、微机、单板机、单片机、掌上电脑、智能芯片等。软件平台指被测软件运行所需要的支持软件,包括操作系统、中文处理系统、数据库系统、工具软件、通信软件、中间件、控制软件、应用软件等。网络平台包括各种网络设备、线缆、连接器件组成的各种结构的网络。其他辅助设备包括各种软件运行和验证其功能而需要的电子、机械、光学等设备或设施。

1. 计算机设备的检查

采用两种方式进行检查。一是采用硬件检测软件对机器硬件进行检查,主要测试项目包括硬盘、内存、CPU、通信端口、主板、显卡、网卡等,具体测试方法参见该软件的使用帮助。如果设备带有专用的检测程序,可以执行其专用的检测程序进行检查;二是查看机器开机时的自检信息,如内存检测、BIOS、硬盘信息、CPU 信息等。

2. 网络设备检查

网络设备通过观察开机自检时的指示灯状态验证网络设备是否正常。通常网络设备状态指示灯自检完成后为绿色,表示正常。

3. 测试用平台软件和支持软件的检查

测试用平台软件和支持软件也是设备检查工作中的一个重要部分。其检查工作主要包括如下内容:

常规检查,主要检查软件的包装是否完整,附件是否齐全。

适用性检查,主要检查软件的类型、版本,确认该软件是否符合测试项目的需要(尽可能保证该软件为原版介质)。

介质检查,主要检查软件介质是否完好,是否能够正常使用。

4. 特殊问题的解决

在测试过程中遇到停电、重新开机等情况时,在机器重新启动过程中,测试人员应仔细观察测试设备的自检过程,以保证测试设备工作正常。

测试设备检查过程中,若发现异常,应及时通知公司相关部门进行处理。总之,所有的测试设备,在进行测试工作前,要保证其可用、可靠。

11.1.2 测试环境搭建

严格来说,参与搭建软件测试环境的所有软硬件设备都需要重新进行安装和调试,以确保测试环境的纯洁,以降低环境对测试结果的影响。

1. 测试设备、设施、软件准备

根据该测试项目的测试计划中列出的设备清单,将所有要用到的硬件、软件和其他设

备准备齐全，并经过检查，确认设备正常并可用后，方可开始测试环境的搭建工作。

2. 设备、设施连接

当所有的软硬件设备和其他相关设备都齐备并确认可用后，可以进行设备、设施的安装、连接。如计算机显示器、键盘、鼠标的连接，网络设备的安装和连接，其他设备的安装和连接等。在硬件设施的安装和连接过程中应该严格按照其安装手册或说明书的要求完成。并遵守一些通用的技术要求，如不要带电操作，要轻拿轻放，安装内部板卡要有防静电措施等。

3. 软件安装

在硬件连接完成后，进行系统软件的安装工作。软件部分的安装包括系统软件安装、支持软件安装、其他测试辅助性软件安装等(但不包括所要测试的软件的安装，它属于测试工作的一部分)。系统安装完成后，如果还需要安装其他支持软件或测试辅助性软件，可一并安装。

4. 调试

调试是在软硬件安装和连接后，为使安装的软硬件设备更好地满足测试的需求而进行的试验和调整，主要包括网络、测试用计算机、其他设备等部分。

网络调试通常使用一些网络工具来验证其连接情况，如 DOS 命令 Ping 等。

测试用计算机的调试主要包括一些显示、通信、存储、设置、服务等情况的验证，如果存在问题，则进行一些修改或调整，以使其工作状态正常。

其他设备的调试可以根据相应的手册进行。

11.1.3 测试环境检查

测试环境是测试执行的保证，如果测试环境设置不对，测试结果就可能存在偏差。因此，在测试执行前，为确保测试环境的工作状态满足测试工作的需要，应对搭建的测试环境进行全面的检查。

测试环境的检查主要分为主机检查、网络检查、其他设备的检查，检查内容分为独立性、正确性及一致性检查。

1. 主机检查

(1) 独立性

主要检查安装软件后的主机的独立性。检查主机是否安装有其他和本次测试无关的软件或程序，如在 Windows 系统中，从“开始”→“程序”菜单中查看系统是否安装了其他软件；检查系统是否安装了和本次测试无关的设备或装置，如在 Windows 系统中打开“控制面板”中的“系统”查看系统硬件信息等；检查操作系统或支持软件是否为新安装的，等等。

(2) 正确性

主要检查软件安装后的主机的功能是否正常。检查主机开机的自检过程和信息是否正常；检查操作系统常用功能是否正常，如存储读写、运算、资源管理等；辅助设备如打印机、扫描仪等能否正常使用；支持类软件能否正常运行，功能是否正常等。

(3) 一致性

主要检查软件安装后的主机是否满足测试的要求。检查主机硬件条件是否满足要求,如主频、内存、板卡等;检查操作系统和支持软件是否满足测试要求,如操作系统类型、版本,支持软件类型、版本,补丁程序类型、版本等;外部设备是否满足测试要求,如打印机类型、扫描仪类型等。

2. 网络检查

(1) 独立性

主要检查网络的独立性。检查网络设备的连接方式是否独立;检查在测试用网络中是否有其他和测试无关的设备,如是否连接了其他和测试无关的主机,是否连接了不需要的路由、通信设备等;当需要和外部连接时,需确保外部连接不会影响测试结果,如检查同外部连接前后测试设备是否正常,软件运行是否正常等。

(2) 正确性

主要检查网络能否正常工作。检查网络连接情况是否正常,可采用 Ping、Telnet 等工具;检查设备配置是否正常,如 VLAN 划分等。

(3) 一致性

主要检查网络是否满足测试的要求。检查硬件设备是否满足测试要求,如传输介质、网络设备型号、品牌、数量等;检查网络架构是否满足测试要求,如设备连接方式等;检查网络配置是否满足测试要求,如 VLAN、路由冗余配置情况等。

3. 其他设备的检查

其他设备或设施的检查也应遵照独立性、正确性、一致性三个方面进行检查,具体检查方法和步骤可以参照其使用手册或维护手册进行。

4. 病毒检测

病毒是软件测试环境检查的一项内容,因为病毒的发作会导致测试结果偏离软件的真实质量状况,得出不公正的结论。另外,在安全性方面,病毒也有可能导致测试结果被非法获取,给委托方造成损害。

11.2 测试执行

测试执行简单来说就是根据测试计划安排执行测试的过程,测试人员参照测试策略,提取测试用例,运用静态分析或者动态测试手段对被测软件系统进行测试。基于用例的测试主要是通过比较分析系统的实际输出结果和期望输出结果,以检验软件系统是否产生正确的输出。如果两者一致,则意味着测试没有发现问题(当然,实际测试期间还要对其他的可能因素进行排查,在很多时候我们还需要对被测系统的数据处理情况进行进一步的验证,比如需要进一步查看软件产品的操作日志、系统运行日志、系统资源使用情况或者后台数据库信息,来判断测试用例是否真的执行成功了);如果两者不一致,则意味着软件系统可能存在错误,需要进一步确认、修改和测试。

11.2.1 测试执行流程

软件测试执行的过程可以抽象为这样一个流程图(如图 11-1 所示),测试执行的输入文档主要是测试计划和测试集(一组测试用例),当然还有安装文档(如果能提供的话),因为我们要据此搭建测试环境。

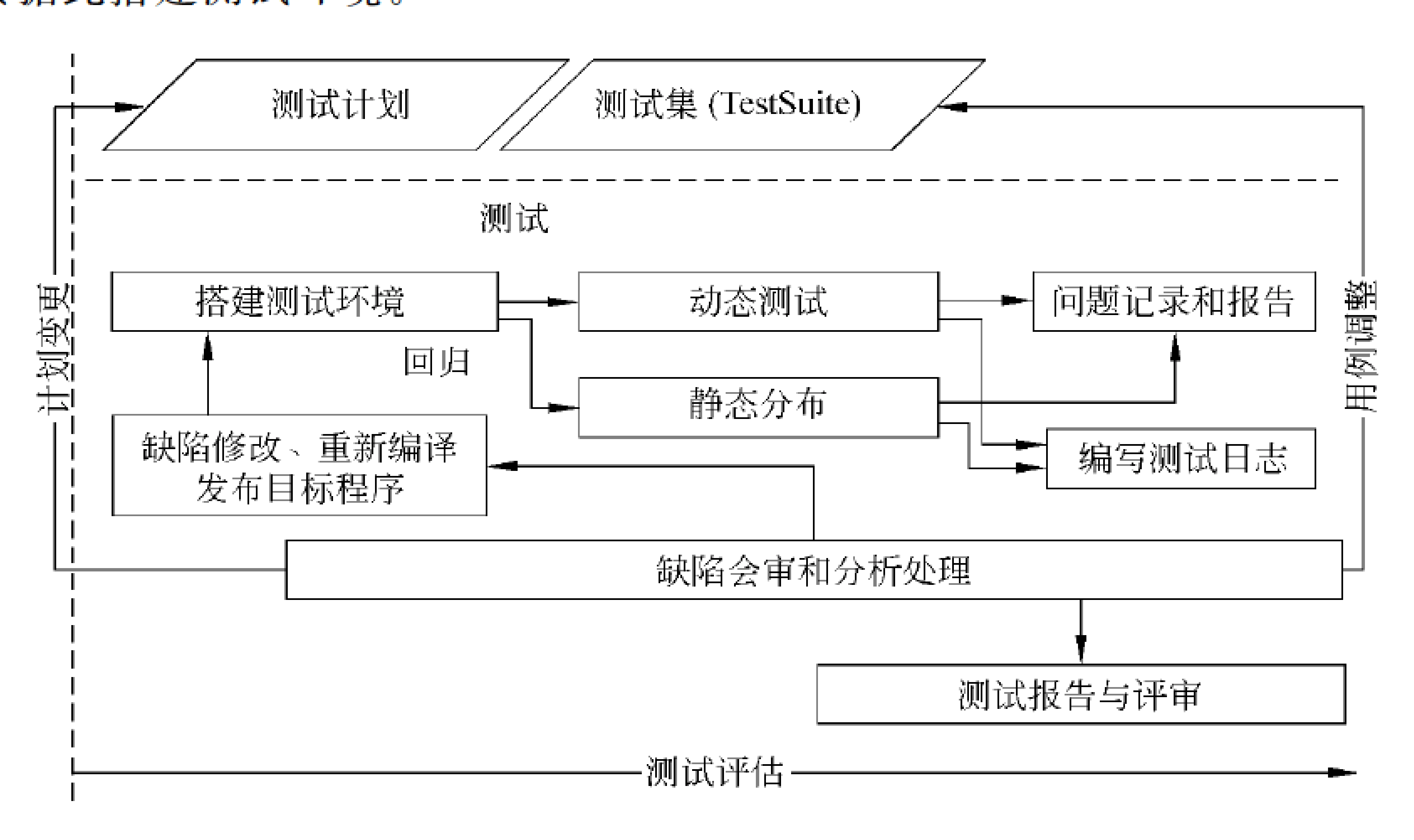


图 11-1 测试执行流程

环境检查确认之后,我们按照测试计划的进度安排,从测试集中选取合适的测试用例进行动态测试和静态分析。其中动态测试按照我们前面的内容介绍可以是功能测试也可以是性能测试,可以是手工模式也可以是自动化模式;静态分析可以用来对代码质量、程序结构等进行度量。

在这些测试活动中,我们要随时保持警惕性,对于实际结果和预期结果不符的情况及时整理,报告差异,这就是缺陷,后面我们会有详细的解释。同时要对测试活动的进行情况,比如执行的进度、偏差、系统状态的变更等编写成日志,与测试组的其他成员和组长共享和沟通,以便于及时识别测试执行中的问题,采取纠正措施。

测试人员会同开发人员共同对存在的缺陷进行确认,必要时要对缺陷进行复现。开发人员对确认的缺陷进行修复,并发布新的测试版本。测试人员(有些企业可能会有专职发布人员)获取新版本并更新测试环境,更新情况要通报给所有成员。测试组适时进行回归测试,直至达到测试预定的退出标准。

11.2.2 监控执行过程

测试执行的过程并不是一帆风顺的,期间将受到很多因素的影响,比如客户需求的变更对开发进度的影响、对系统设计的影响等,这些会直接作用于测试执行过程,导致测试进度的调整、用例的失效等。因此,在测试执行期间,我们要对测试过程进行必要的和适度的监控,以及时发现执行期间出现的问题及形成因素,并对这些问题做出评估,提出解决办法,保证测试活动的顺利开展。

测试监控目前来说尚处于一种实践和探索的阶段，还没有建立起统一的理论体系，我们在本节向大家介绍的也是基于实践经验的总结。测试监控的强度和要求对于不同的测试组织可能是不一样的，比如对于质量管理体系健全完善的组织来说，可能监控的范围会更广一些，可以扩展到测试成员的测试日志及日志审核情况、质量会议的召开或者参与情况等。另外，测试监控的要求随着监控人的目的不同也是有差异的，比如测试组长和组员、测试管理者和测试组长等，随监控目的的不同，他们会呈现出不同的监控广度和监控深度。虽然如此，一般来说，测试执行期间最基本的监控点有四项，分别是执行进度、测试用例执行情况、版本控制和缺陷管理情况（如图 11-2 所示）。

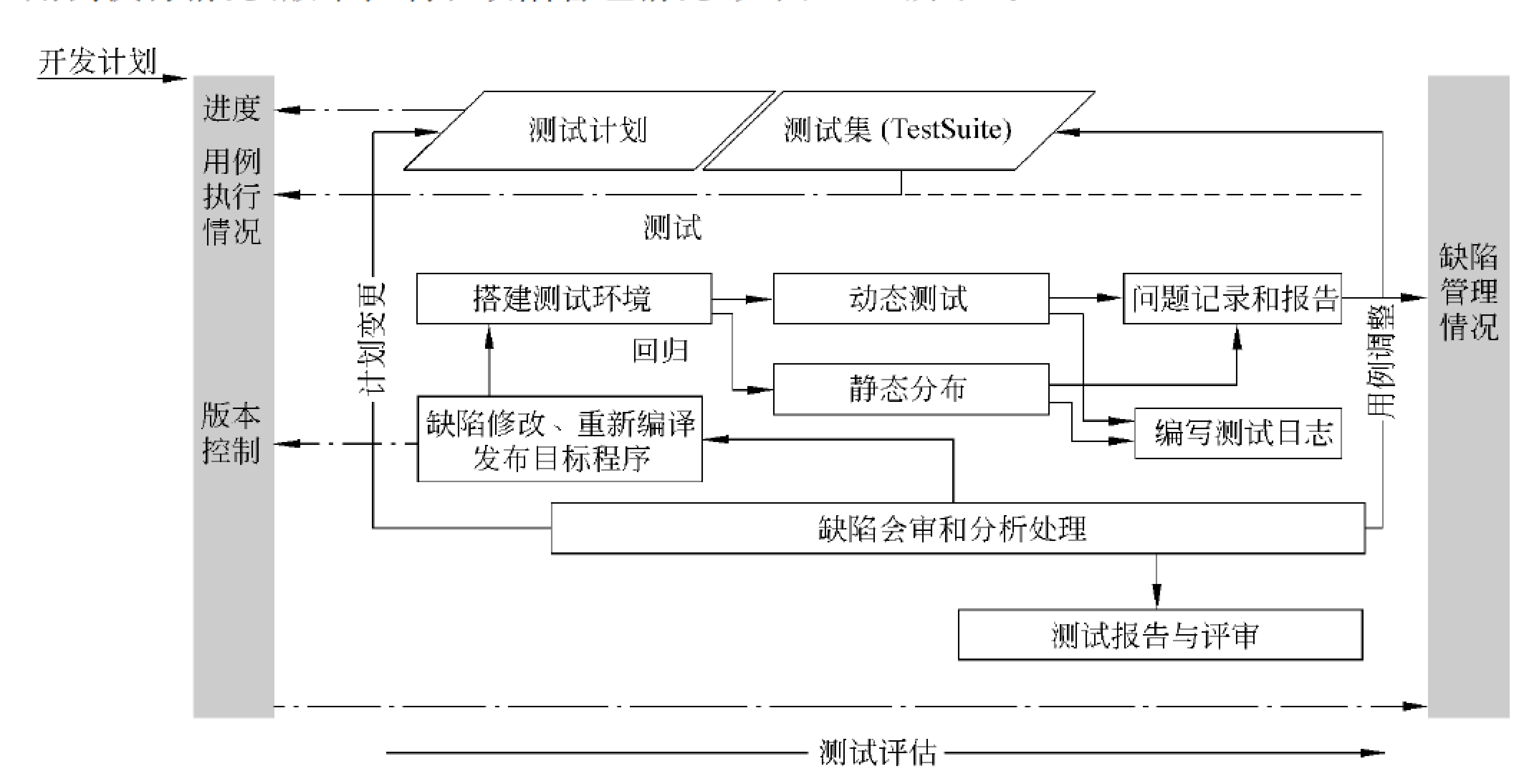


图 11-2　执行过程监控

1. 执行进度

执行进度是测试执行期间首先要关注的问题，要实施掌握测试进程是不是按照测试计划中预期的时间安排进行的。如果存在延期现象则要纠察原因，做出修正或者调整，以保持整体测试进度，避免局部的变化对整体测试进度的影响，必要时还要做出风险评估，以风险管理的手段进行变更控制。这是对于负面变更的处理方式，对于正面的变化，比如某些测试项提前完成了，那么测试负责人必要时可以对资源重新调配，对于测试权重大的部分可以投入更多的兵力。

以上是对于测试本身的进度变化情况的监控和处理，另外我们在测试计划阶段已经了解到开发计划是影响测试计划的一个直接因素，因此在测试执行期间对于开发计划的变更也要给予足够的关注，并对测试计划的影响做出必要评估，并制定出合适的应对方案。

开发计划的变化无非有提前、延期和废止这样几个方面，对应于这几个方面，测试负责人可以对测试计划做出如下调整：

(1) 将原定于后期测试的测试项提前，当然这得在满足测试项预置条件的前提下；对于有些软件项目，虽然部分开发项提前完成了，但是这些项在原测试计划中并没有安排独立测试，而是和其他测试项集成后进行测试，则这些就不会对测试计划造成影响。

（2）部分开发任务延期，我们要分析出与该任务关联的所有待测试项，如延期开发的是一个公共组件，则所有调用这个组件的开发项都会被迫延期，则对应的所有测试项也将不得不延期。

（3）遇到开发任务废止，也是要分析出与该任务关联的所有待测试项，并将这些测试项从测试计划中摘除，通报测试小组即可。

我们以上所说的都是在排除了一些额外条件之后的处理方式，对于有些质量管理体系健全的企业，测试人员可能还被赋予了更多权利，会参与到这些开发计划的变更中，并代表最终用户参与部分意见，从测试的角度分析计划变更对系统整体质量的影响，提出自己的评估结果和建议，从而对计划变更的程度施加自己的影响。

2. 用例执行情况

用例执行情况是对测试设计的监控。通过对测试用例执行情况的监控，可以检验测试用例设计的质量、测试用例的执行效率等。

测试用例的质量可以通过以下两个指标进行考查：

1）测试用例检出缺陷率

测试用例检出缺陷率（UCR_0）可以通过以下公式计算：

$$UCR_0 = S/T \times 100\%$$

其中，S 是测试未通过且确认了系统缺陷的用例数量，T 是全部测试用例。

2）用例有效率

用例有效率指可以完全按照测试用例执行，与系统设计一致的用例占全部用例的比率。这个可以通过以下公式进行计算：

$$UCR_1 = S/T \times 100\%$$

其中，S 是未做变更的用例数，T 是全部测试用例。

另外，所准备的测试用例是否足够，即测试用例的覆盖度可以通过统计缺陷漏检率来进行考查。

考查这个指标可以有两种计算方式，一种是粗线条的，公式如下：

$$\text{缺陷用例比} = \text{模块缺陷数}(D)/\text{模块用例数}(UC) \times 100\%$$

缺陷用例比越高表示测试用例越不充分。缺陷漏检率是一种精细化的计算方式，即可以具体统计一个模块发现的缺陷哪些是测试用例执行期间发现的（D_0），哪些是非执行测试用例发现的（D_1），然后据此可以计算缺陷漏检率如下：

$$\text{缺陷漏检率} = D_1/(D_0 + D_1)$$

在测试执行期间通过对测试用例执行情况的监控，可以及时发现测试用例存在的问题，改进软件测试设计过程，提高测试用例的质量，使测试人员和开发人员提升对测试用例的信心；对于漏检比较多的模块，可以及时补充更多的测试用例，覆盖更多的功能项。这就是监控测试用例执行情况的基本目的和作用。

除了上述这些测试用例执行情况监控的工作之外，事实上测试用例执行情况的监控对于测试执行质量也有很好的促进作用。

现在问大家一个问题：在测试执行过程中，当实际输出结果与测试用例中的预期结果一致的时候，是否就可以认为测试用例执行成功了呢？这对于有经验的测试人员当然知道，而对于一个新手来说恐怕要给出肯定的答案来了，事实上很多时候不能够简单地依

据这个条件去给出想当然的判断，在很多时候我们还需要对被测系统的数据处理情况进行进一步的验证，比如需要进一步查看软件产品的操作日志、系统运行日志、系统资源使用情况或者后台数据库信息，来判断测试用例是否真的执行成功了。所以说，适度的对测试用例情况进行监控（比如抽检、复检）可以及时发现和纠正测试人员在测试执行期间出现的问题。

3. 版本控制

版本控制（Version Control）是一种软件工程技巧，借以在开发的过程中，确保由不同人员所编辑的同一档案都得到更新，同时又可以方便地回溯到档案任何一个历史版本。

版本控制透过文档控制（Documentation Control）记录程序各个模组的改动，并为每次改动编上序号。这种方法是工程图维护的标准做法，它伴随着工程图从图的诞生一直到图的定型。例如，赋给图的初版一个版本等级“A”。当做了第一次改变后，版本等级改为“B”，依此类推等等。

版本控制在大型应用软件开发过程中得到了广泛的普及和应用，透过版本控制系统使测试人员和开发人员也得到了圆满的协同。版本控制可以确保测试人员正确地得到软件的最新发布版本，使测试环境得到有序的更新。对版本控制的监控就是要确保授权人员合理地使用版本控制系统，根据软件测试的进度适时对测试环境的目标测试系统做出正确更新。

实际上，随着软件测试的发展，软件测试自身也产生了版本控制的要求。比如测试用例会伴随着测试进程不断地发生变化，每一次变化是事实上的一个测试版本，我们需要对这些版本进行统一的管理，以确保测试组成员正确地获得所需要的测试用例，这就是测试用例的版本控制。

测试配置管理是软件配置管理的一个组成部分，贯穿整个软件测试生命周期。将软件测试纳入配置管理一是可以提高软件测试过程文档的共享程度，使测试信息更加透明，另外可以降低软件测试过程中产生的软件变更引起的错误放大风险，可以像数据库管理一样方便地实现管理对象的“回滚”。测试配置管理的实施过程一般是：

1）配置项识别

配置项顾名思义就是需要纳入配置管理系统进行版本控制的对象。测试配置管理的管理对象对应于软件测试生命周期，可以有测试计划、测试用例（或自动化测试脚本）、测试驱动模块（桩模块）、测试日志、缺陷记录和测试结果等，其他的可能还有一些重要的会议纪要等内容。

2）版本控制

对识别出的配置项进行版本标记，以区别配置项的变更。版本控制一般采用专门的配置管理工具实现。

3）变更控制

一般情况，配置管理员会把评审过的重要配置项转入基线库，作为以后任何变更的记录和参考依据，任何对基线版本的变更都需要经过预先定义的流程来加以控制。我们把这个过程称为变更控制。变更控制的目的并不是控制变更的发生，而是对变更进行管理，确保变更有序进行。

4. 缺陷管理

缺陷管理是测试执行期间监控的一个核心内容。缺陷管理的监控主要有以下方面：

1）缺陷录入情况

缺陷录入包含了较多测试人员的主观因素，因此对缺陷录入的监控是十分必要的。把好缺陷录入的关口，尽可能降低无效缺陷的录入，可以节约测试成本。

对于缺陷录入情况，可以采取缺陷审查、交叉确认等形式。测试负责人可以召集测试组成员快速地对所录入缺陷进行审查，对缺陷的描述、状态的设置和测试步骤等进行审核，确认是否符合编写要求，这种形式称之为缺陷审查；测试负责人也可以将已录入缺陷按照隔离原则，将缺陷分发给不同的测试人员进行复检，以对缺陷进行确认，这种方式称为交叉确认。

2）缺陷确认情况

简单地说缺陷确认后缺陷被分化为要修改的和不修改的（关于缺陷状态的设置和变化会在 11.3 详细说明），对于不修改的部分我们应该予以更多关注，要及时对不修改的缺陷做分析，防止遗漏真正的缺陷。

另外，缺陷有效率是缺陷确认情况的一个指标，它反映了录入缺陷的准确度。因此，我们要统计缺陷有效率这样一个指标，具体计算公式如下：

缺陷有效率＝已确认真实缺陷数/缺陷总数×100％

缺陷有效率反映了测试人员发现缺陷的质量，缺陷有效率越低说明测试人员所提交的缺陷质量越差，存在较多误判缺陷。如果出现这种情况，测试负责人要及时查明原因，做出纠正措施。

缺陷确认率是缺陷确认情况的另一个指标，它反映的是开发人员对缺陷的响应程度，缺陷确认率越高说明开发人员对缺陷响应越及时，反之则说明开发人员对缺陷响应越迟钝。如果出现这种情况，测试负责人要及时跟项目经理或者开发负责人进行沟通，查明原因，以适时对测试进度做出必要调整。缺陷确认率的计算公式可以表示如下：

缺陷确认率＝已确认缺陷数/缺陷总数×100％

3）缺陷修改情况

缺陷修改情况可以考查缺陷修改率，缺陷修改率反映了开发人员对于缺陷的处理进度，这个指标越高，说明开发人员处理问题越及时，反之则说明越迟钝。缺陷修改情况的计算公式表示如下：

缺陷修改率＝已修改缺陷数/已确认真实缺陷总数×100％

4）缺陷验证情况

测试人员要对开发人员修改过的缺陷再次进行测试，以验证缺陷是否已经被纠正，另外还要对受此影响的部分进行测试，已确认变更没有影响到原有系统功能。这个过程称之为回归测试。监控缺陷验证情况可以准确了解回归测试的进度。缺陷验证情况的指标可以按照以下公式进行设定：

缺陷验证率＝已验证缺陷/已修改缺陷总数×100％

11.2.3 测试执行记录

测试执行过程的好坏直接影响整个测试计划的实施效果，因此对于软件测试执行过程的管理尤其显得重要。为了保证软件测试的有效性和一旦出现问题后的可跟踪、可复现，软件测试在执行期间应保持文档的完整性和连续性。本节将按照测试执行的顺序，结

合 GB/T 15532—2008《计算机软件测试规范》和 GB/T 9386—2008《计算机软件测试文档编制规范》这两个最新标准规范的要求介绍一下软件测试执行期间所需要编制的文档。

1. 测试项传递报告

在搭建测试环境期间并没有包括被测系统的安装和调试。因此，在开始测试前，我们首先要把被测系统安装到已准备的测试环境中。这个过程中，被测系统作为目标测试项将从开发组织转移到测试组织，如果是第三方测试机构，被测系统将从委托测试方转移到被委托方，这种转移称为“软件移交”（软件移交的有关内容可参考第 8 章）。这种移交是一种正式行为，为了避免因此产生的纠纷，一种好的做法是对这种移交进行记录。在 GB/T 9386—2008《计算机软件测试文档编制规范》中，这种记录叫做“测试项传递报告”。

测试项传递报告用以指明在开发组和测试组独立工作的情况下或者在希望正式开始测试的情况下为进行测试而传递的测试项。它的主要内容包括以下 5 个方面：

1）传递报告标识符

为该测试项传递报告规定唯一的标识符。

2）传递项

标识被传递的各个测试项，其中包括其版本/修订级别。提供对传递项有关的测试项文档集和测试计划的引用，并指出负责该传递项的人员。传递项中的文档集如果存在电子版，则应指明其版本和存放路径。

3）位置

标识各传递项的位置，并标识包含被传递项的媒体。适当时，指出标记或者标识特定媒体的方法。

4）状态

描述被传递项的状态，包括与该测试项文档集、与这些测试项的先前传递以及与测试计划的偏离。列出期望由被传递项解决的各个事件报告，指出是否存在对测试项文档集的未决的修改，该修改可能影响在该传递报告中列出的各个测试项。

5）批准

详细说明最可能批准该传递项报告的人员姓名和职位，并为签名和日期留出位置。

【应用实例：HRMIS 传递项报告】

1. 传递报告标识

传递报告标识符：HR01-01 报告日期：20××年 12 月×日

2. 传递项

“人力资源管理系统”的 V0.2 安装程序，存放位置：1 号版本控制服务器

《需求规格说明书》纸质 1 份，电子版存放位置：1 号版本控制服务器

《设计规格说明书》纸质 1 份，电子版存放位置：1 号版本控制服务器

3. 位置

HRMIS 发布程序为可执行文件 hrmis.exe 及支持库，可从 1 号版本控制服务器上检出。

4. 状态

HRMIS 已经通过单元测试和集成测试，准备进行系统测试。

5. 变更说明

该版本用于解决20××年12月×日形成的一个事件报告(HR03-01)。

程序的“培训管理”模块,输入项“主题”缺乏合法性校验导致事件(HR03-01),现已增加了校验。

6. 批准

开发经理:　　　　日期:

测试经理:　　　　日期:

2. 测试日志

测试日志主要用于记录测试执行过程中发生的情况。为了保证测试的有效性,出现问题时能够及时捕捉和复现问题,测试组应养成编写测试日志的良好习惯。在GB/T 9386—2008《计算机软件测试文档编制规范》中,规定测试日志一般包含以下内容:

1)测试日志标识符

为该测试日志规定唯一的标识符。

2)描述

标识被测试的各个测试项及其版本/修订级别。对于其中的每一项,如果存在其传递报告,则应加以引用。

标识执行测试的环境属性,包括设置说明、使用的硬件(例如内存容量、CPU型号、硬盘型号容量等)、使用的系统软件及可用资源。

3)活动和事件条目

对于每个事件,包括事件的开始和结束,要记录发生的日期和时间以及记录者的身份。具体包括:

- 执行描述:记录正在执行的测试规程的标识,提供并引用该测试规程说明。记录执行时在场的所有人员,包括测试者、操作者、观察员,还要说明每个人的职能。
- 测试结果:对每次执行,记录目视可观察到的结果(例如产生的出错消息、异常终止和对操作员动作的请求等),还要记录任何输出的位置(例如磁带编号),以及记录测试的执行是否成功。
- 环境信息:记录与本条目有关的任何特殊的环境条件(例如硬件更换)。
- 异常信息:记录某个不期望事件(例如:某个操作的系统响应时间过长)发生的前后的情况。
- 事件报告标识符:随时记录每个测试事件报告产生的标识符。

【应用实例:HRMIS系统测试日志】

1. 测试日志标识

测试日志编号:HR02-01　　日期:20××年12月×日

2. 描述

“人力资源管理系统”程序的V0.2版本正在测试。

正在利用标准的公司检测室的设施来进行测试。

该日志记录“培训管理”功能模块的测试执行情况。

3. 活动和事件条目

20××年 12 月×日

下午 2:00　王强 开始测试

下午 2:15　在测试“增加培训信息”时,系统异常退出。

形成事件报告 HR03-01(事件报告标识符)

填好事件报告并围绕出现的问题协助开发人员进行缺陷复现和原因分析

下午 5:00　完成“删除培训信息”测试的数据准备工作

下午 5:15　王强 停止测试

20××年 12 月×日

上午 9:00　王强 开始测试

上午 9:30　完成“机构管理”功能的测试,测试用例全部通过。

上午 10:00　王强 停止测试

上午 11:00　张伟 开始测试

下午 14:00　在测试“培训机构管理”时发现一个可能存在的问题。

形成事件报告 HR03-02

下午 16:00　张伟 停止测试

3. 测试事件报告

在 GB/T 9386—2008《计算机软件测试文档编制规范》中还定义了一种文档,叫做测试事件报告。测试事件报告是一类用以记录在测试执行期间发生并需进一步调查的任何事件的记录文档。测试事件报告的一般编写要求如下:

1) 测试事件报告标识符

为测试事件报告规定唯一的标识符。

2) 摘要

概述事件。标识所涉及的所有测试项,指出其版本/修订级别。应提供对有关测试规程说明、测试用例说明和测试日志的引用。

3) 事件描述

记录并报告测试中出现的异常现象或者人为事件。事件描述应包括以下内容:输入、预期结果、实际结果、异常现象、日期和时间、规程步骤、环境、重复执行的意图、测试者和观察者。

事件描述应包括可能有助于隔离并纠正事件原因的相关活动和观察结果。例如:描述可能对此事件以后影响的所有测试用例执行情况,描述与已公布的测试规程之间的任何偏差。

4) 影响

在所知道的范围内指明该事件对测试计划、测试设计说明、测试规程说明或测试用例说明所产生的影响。

【应用实例：HRMIS 测试事件报告】

1. 测试事件报告标识

报告编号：HR03-01　　报告日期：20××年 12 月××日

2. 概述

“人力资源管理系统”中，执行“增加培训信息”时，输入符合要求的信息，无法成功添加记录。

该事件发生在执行测试用例 P200901ST-PXGL-PXLR-××的期间。测试日志 HR02-01 记录了该事件。

3. 事件描述

20××年 12 月××日　　下午 2:15 王强

执行“增加培训信息”功能，各输入项分别输入符合数据字典要求的数据，然后执行保存，系统提示“CADORecordset Error”错误，执行确定后，程序异常退出。

4. 影响

在解决该事件之前暂停测试活动。

测试事件报告适用于测试设计(分析)和测试执行完全分离的测试组织，在这类组织中测试执行人员自身不足以对测试期间捕捉到的现象进行缺陷判断时需要将完整的事件信息呈送给测试设计(分析)人员，他们之间的这种信息传递中介就是测试事件报告。

测试事件报告不能等同于问题(缺陷)报告。我们知道，测试用例的执行结果有两个，一个是 Pass，一个是 Failure，对于 Failure，测试执行人员作为事件整理并报告给测试设计(分析)人员，测试设计(分析)人员对其所报告事件进行分析处理。分析处理的结果可能是缺陷，但是也可能是由于其他因素导致的“伪缺陷”，比如环境造成的错误、人为因素造成的错误或者用例本身设计错误等(如图 11-3 所示)。

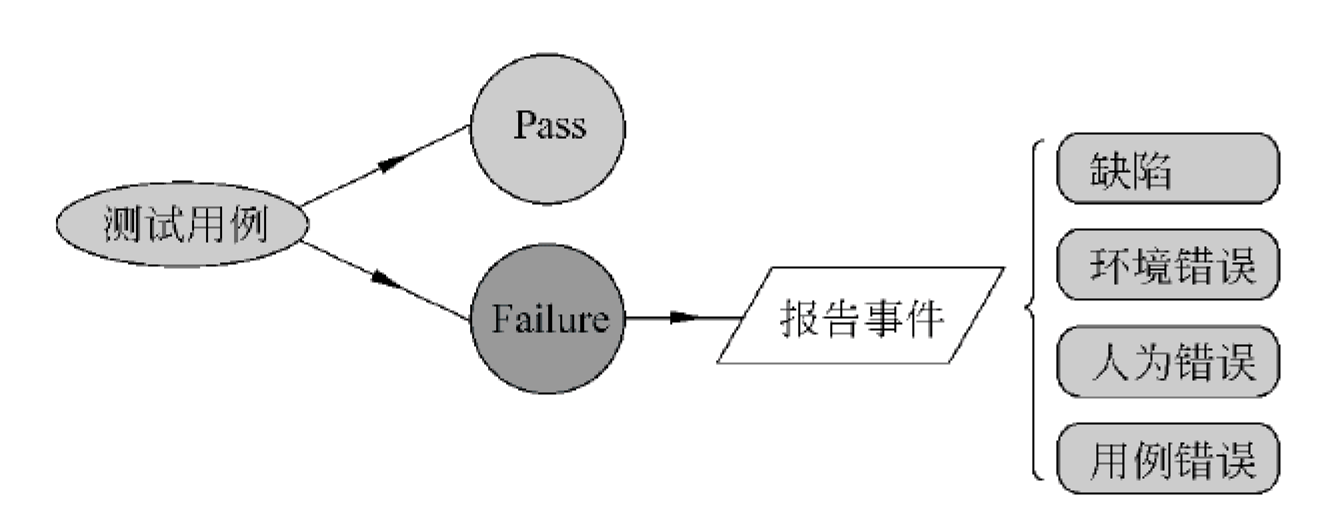


图 11-3　事件报告

4. 测试记录与软件问题报告单

在测试执行过程中，测试人员应认真观察并如实地记录测试过程、测试结果和发现的错误。测试员应详细记录每个测试步骤的实测结果，不管是与预期输出相符的或不符的结果都必须作记录，也就是说，测试记录必须包括所有测试项的执行结果；当实际执行结果与预期输出相同时，可在实际输出处注明与预期输出一致；当出现错误时，按照记录的执行过程应能重现该错误。对于那些应该测试，但在测试时，由于某种原因不作测试的测试项，在对应的“实测结果”栏注明“未测试项”，并说明此项不作测试的原因，这就是测试

记录。在 GB/T 15532—2008《计算机软件测试规范》中，规定测试记录具有如下格式。

<table>
<tr><td colspan="2">用例名称</td><td colspan="1"></td><td>用例标识</td><td></td></tr>
<tr><td>用例说明</td><td colspan="4"></td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="3"></td></tr>
<tr><td>软件配置</td><td colspan="3"></td></tr>
<tr><td>测试配置</td><td colspan="3"></td></tr>
<tr><td>参数设置</td><td colspan="3"></td></tr>
<tr><td colspan="5">操作过程</td></tr>
<tr><td>序号</td><td>输入及操作说明</td><td>期望的测试结果</td><td>评价标准</td><td>实测结果</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td colspan="2">是否发生重启动 □</td><td>重启动是否成功 □</td><td>是否发生失效 □</td><td>是否发生故障 □</td></tr>
<tr><td colspan="2">测试结论</td><td colspan="3"></td></tr>
<tr><td colspan="2">测试人员</td><td></td><td>测试日期</td><td></td></tr>
</table>

测试事件报告不能等同于缺陷报告，还需要进一步对事件进行分析才能确认。测试人员根据每个测试用例的期望测试结果、实际测试结果和评价准则判定该测试用例是否通过，并将结果记录在软件测试记录中。如果测试用例不通过，测试人员应认真分析情况，采取相应措施。常见的测试用例不通过的情况及其处理措施包括：

- 测试说明和测试数据的差错。采取的措施是：改正差错，将改正差错信息详细记录，然后重新运行该测试。
- 执行测试步骤时的差错。采取的措施是：重新运行未正确执行的测试步骤。
- 测试环境（包括软件环境和硬件环境）中的差错。采取的措施是：修正测试环境，将环境修正情况详细记录，重新运行该测试；若不能修正环境，记录理由，再核对终止情况。
- 系统实现的差错。采取的措施是：填写软件问题报告单，可提出软件修改建议，然后继续进行测试；或者把差错与异常终止情况进行比较，核对终止情况。软件变更完毕后，应根据情况对其进行回归测试。
- 系统设计的差错。采取的措施是：填写软件问题报告单，可提出软件修改建议，然后继续进行测试；或者把差错与异常终止情况进行比较，核对终止情况。软件变更完毕后，应根据情况对其进行回归测试或重新组织测试，回归测试中需要相应地修改测试设计和数据。

在 GB/T 15532—2008《计算机软件测试规范》中，软件问题报告单的编写格式如下：

<table>
<tr><td colspan="2">问题标识</td><td></td><td>项目名称</td><td></td><td>程序/文档名</td><td></td></tr>
<tr><td colspan="2">发现日期</td><td></td><td>报告日期</td><td></td><td>报告人</td><td></td></tr>
<tr><td rowspan="2">问题性质</td><td>类别</td><td>程序问题 □</td><td>文档问题 □</td><td>设计问题 □</td><td>其他问题 □</td><td></td></tr>
<tr><td>级别</td><td>1 级 □</td><td>2 级 □</td><td>3 级 □</td><td>4 级 □</td><td>5 级 □</td></tr>
<tr><td colspan="2">问题追踪</td><td colspan="5"></td></tr>
<tr><td colspan="7">问题描述/影响分析：(可另加附页)</td></tr>
<tr><td colspan="7">附注及修改建议：(可另加附页)</td></tr>
</table>

5. 测试总结报告

每一个测试项完成测试后，测试人员需要对该软件测试项的测试情况进行总结，在GB/T 9386—2008《计算机软件测试文档编制规范》中这种总结性文档称之为测试总结报告。测试总结报告是用来总结测试活动和结果的文档，在软件测试生命周期中算是比较正式的文件，它的一般编写格式是这样的：

(1) 测试总结报告标识符

为测试总结报告规定一个唯一的标识符。

(2) 摘要

总结对测试项的评价。标识已经测试的各个测试项，指出其版本或修订级别，并指出执行测试活动所处的环境。

对于每个测试项，应写明所引用的测试文档。

(3) 差异

指出测试项与其对应的设计说明之间的任何差异，并指出与测试计划、测试设计之间的任何差异，同时详细说明每种差异产生的原因。

(4) 测试充分性评价

应根据测试计划中规定的测试充分性准则对测试过程作出评价。若测试不充分，应指出哪些方面未做充分测试，并说明理由。

(5) 结果汇总

汇总测试的结果。列出已经解决的所有事件，并总结其解决方案。并指出尚未解决的所有事件。

(6) 评价

对每个测试项进行总体评价。评价必须以测试结果和测试项级别的通过准则作为依据。可以包含对失败风险的估计。

(7) 活动总结

总结主要的测试活动和事件。总结资源消耗数据，例如，人员的总体配备水平、每个主要测试活动所花费的时间。

(8) 批准

详细说明必须批准该报告的所有人员的姓名和职务，并为签名和日期留出位置。

【应用实例：HRMIS 系统测试总结报告】

人力资源管理系统测试总结报告

1. 测试总结报告标识符

ST-0901-001-RPT

2. 摘要

组成公司人力资源管理系统的所有功能项，包括员工管理、培训管理、系统设置三个模块均通过了所有测试。

下列文档用作本测试总结报告的信息源：

- 人力资源管理系统(HRMIS)测试计划；
- 人力资源管理系统(HRMIS)测试用例；
- 人力资源管理系统(HRMIS)测试记录；
- 人力资源管理系统(HRMIS)缺陷报告。

3. 差异

软件的规格说明中未对“岗位设置”功能进行描述，结果产生了另外 10 个测试用例，所有这些更改均包含在当前的测试用例集中。

4. 测试充分性评价

本次系统测试充分，已达到测试计划中规定的测试充分性准则要求。

5. 结果汇总

测试用例中的两个用例(用例标识为 P200901ST-PXGL-PXLR-××、P200901ST-PXGL-PXLR-××)暴露了由于“员工新增”功能缺少合法性校验导致系统非法退出的故障。完善该功能后，重新运行测试集，所有用例都通过测试。

6. 评价

测试项	项目说明	结果
员工管理	主要包括员工信息的增加、修改、删除、查询、培训安排，以及合同管理、工作经历管理等功能	通过
培训管理	主要包括培训信息的增加、修改、删除、查询等功能	通过
系统设置	主要包括部门设置、用户管理、岗位设置等功能	通过

7. 活动总结

开始测试：××-××-××	估计时间	实际时间
测试设计(包括用例)	5 天	6 天
执行测试	4 天	3 天
缺陷修改	1 天	1.5 天
回归测试	1 天	1 天
测试报告	0.5 天	0.5 天
结束测试：××-××-××	总计 12.5 天	总计 12 天

8. 批准

测试经理：　　　　　　　　　　日期：

11.3 缺陷管理

11.3.1 什么是缺陷

软件测试过程中常会出现错误、故障、缺陷、失效等术语，对于软件测试人员，正确区分这些术语的概念尤为重要，它关系到测试人员对软件失效现象与机理的理解。

1. 软件错误

John D. Muse 对"软件错误"的定义是："软件错误是代码中的缺陷，是由错误引起的，是由一个或多个人的不正确或遗漏行为造成的。"是指在软件生存周期内的不希望或不可接受的人为错误，相对于软件本身，是一种外部行为，其结果是使软件产生缺陷。

2. 软件缺陷

指存在于软件（文档、数据、程序）中那些不希望或不可接受的偏差，其结果是软件运行于某一特定条件时出现软件故障，这时称软件缺陷被激活。2001 年，美国 Ron Pattern 在《软件测试》一书中说明，只要软件出现的问题符合以下任一情况，就叫做软件缺陷：

① 软件未达到产品说明书中标明的功能。

② 软件出现了产品说明书中指明的不会出现的错误。

③ 软件功能超出了产品说明书指明的范围。

④ 软件未达到产品说明书中指明的应达到的目标。

⑤ 软件测试人员认为软件难以理解、不易使用、运行速度慢，或最终用户认为不好使用。

3. 软件故障

GB/T 16260—2006《软件工程　产品质量》(idt：ISO/IEC 9126)中定义"软件故障"为"计算机及程序中不正确的步骤、过程或数据定义"。是指软件运行过程中出现的一种不希望或不可接受的内部状态，是一种动态行为。当软件故障无适当的措施加以处理，便产生软件失效。

4. 软件失效

GB/T 16260—2006《软件工程　产品质量》(idt：ISO/IEC 9126)中定义"软件失效"为"产品完成所需功能的能力的终止，或在原先规定的限制内没有能力完成"，是指软件运行时产生的一种不希望或不可接受的外部行为结果，是系统行为对用户的要求，是一种面向用户的概念。

综上所述，软件错误导致软件缺陷，一个软件错误可能产生一个或多个软件缺陷；软件缺陷被激活时，便产生了软件故障，同一软件缺陷在不同条件下被激活，可能产生不同的软件故障；软件故障没有采取正确措施及时处理，便不可避免地导致软件失效，同一软

件故障在不同条件下也可能产生不同的软件失效。

软件缺陷是对软件产品预期属性的偏离现象，指程序中存在的错误，也指存在于设计、需求、规格说明或其他文档中的错误，例如语法错误、标点符号错误或者是一个不正确的程序语句，是任何影响程序完整而有效地满足用户要求的地方，是可以表示、描述和统计的客观事物。

11.3.2 缺陷分类

缺陷是由一组属性组成的，包括缺陷标识、缺陷类型、缺陷严重程度、缺陷优先级、缺陷状态、缺陷起源、缺陷来源、缺陷根源等。

- 缺陷标识：标记某个缺陷的一组符号，每个缺陷必须要有一个唯一的标识。
- 缺陷类型：根据缺陷的自然属性划分的缺陷种类，如系统缺陷、数据缺陷、数据库缺陷、接口缺陷、功能错误、界面错误等。
- 缺陷严重程度：因缺陷引起的故障对软件产品的影响程度，如致命的、严重的、一般的、次要的等（见表 11-1）。

表 11-1 缺陷严重程度分类

描述	等级	描　　述	类　　型
致命的	一级	不能执行正常工作功能或重要功能，使系统崩溃或资源严重不足的情况	• 由于程序所引起的死机，非法退出 • 死循环 • 数据库发生死锁 • 错误操作导致的程序中断 • 严重的计算错误 • 与数据库连接错误 • 数据通信错误
严重的	二级	严重地影响系统要求或基本功能的实现，且没有办法更正的情况（重新安装或重新启动该软件不属于更正办法）	• 功能不符 • 程序接口错误 • 数据流错误 • 轻微数据计算错误
一般的	三级	严重地影响系统要求或基本功能的实现，但存在合理的更正办法的情况（重新安装或重新启动该软件不属于更正办法）	• 界面错误（附详细说明） • 打印内容、格式错误 • 简单的输入限制未放在前台进行控制 • 删除操作未给出提示 • 数据输入没有边界值限定或不合理
次要的	四级	使操作者不方便或遇到麻烦，但不影响执行工作或功能实现的情况	• 辅助说明描述不清楚 • 显示格式不规范 • 系统处理未优化 • 长时间操作未给用户进度提示 • 提示窗口文字未采用行业术语
	五级	建议及其他错误等	

- 缺陷优先级：指缺陷必须被修复的紧急程度，如立即修复、正常排队等待修复、方

便时进行纠正等；

- 缺陷状态：指缺陷通过一个跟踪修复过程的进展情况，如提交、等待处理、拒绝修复、已修复、关闭等；
- 缺陷起源：指缺陷引起的故障或事件第一次被检测到的阶段，如需求阶段、架构阶段、设计阶段、编码阶段、测试阶段等；
- 缺陷来源：指引起缺陷的起因，如需求的问题、架构的问题、设计的问题、编码的问题、测试的问题、集成的问题等；
- 缺陷根源：指发生错误的根本因素。

11.3.3 缺陷报告编写

如果我们有一定的测试工作经历，应该遇到过这样的情景：在某次缺陷评审会议上，开发人员甲对于一个缺陷处理的意见是"reject"，测试人员乙据理力争，认为这个问题确实存在而且对系统造成的损害也是显著的。正当两人争得面红耳赤时，QA 出来打圆场，说不如到测试环境中去实地看看吧，于是尴尬的场景出现了：测试人员乙无论如何再也捕捉不到那个该死的 Bug 了。为什么呢？因为乙留下的测试信息实在是太少了，他已经记不清自己当时的测试步骤和测试数据了，这个问题无法得到复现当然也就得不到修正。

在这个例子中，也许这个问题是确实存在的，也许带着这个缺陷的系统也给未来的应用埋下了隐患，但是因为缺陷报告叙述不充分导致这个问题最终无法得到修正，成了一种擦肩而过的遗憾。

这个教训是显而易见的，这就要求我们要严格要求自己，缺陷报告的编写马虎不得。那么如何编写缺陷报告呢？有人总结出了"信、雅、达"，也就是说缺陷报告的叙述要做到信、雅、达，所谓信当然是指诚信，缺陷报告要做到真实描述，不误报、不妄报，有则有之，无则无之，不要让一些"莫须有"的缺陷浪费掉宝贵的测试时间；所谓雅，就是要求测试报告的描述要简洁清楚，使接受缺陷报告的人能够很快理解你所报告的缺陷；所谓达，就是要详尽，把缺陷需要复现的必要信息都要写清楚，比如测试的步骤、所使用的数据等。下面介绍几个缺陷报告的编写技巧。

1. 组织与复现

测试人员应该尽量采用测试用例执行测试，这样可以保证测试的有效性和记录的完整性，对于随机测试要做详尽的记录。测试人员在编写缺陷报告前必须要对问题的可复现性进行确认。如果错误不是每次都能重现，则一个好的做法应该是对测试的步骤和数据特点进行分析，以查明是测试用例的问题，还是系统其他关联功能的问题，如果实在找不出问题所在，那么在报告缺陷时，应指明这个缺陷的出现频率。

2. 隔离和归纳

在尝试编写缺陷报告之前，可以试着隔离错误。比如采用改变一些变量的方法，如系统的配置，它可能会改变错误的症状。这些信息也许可以为开发人员着手调试提供更多的思路。在测试人员发现了一个已隔离的，可重现的问题后，应该对问题进行归纳。试着做以下方面的思考：同一个问题是否出现在其他的模块或其他的地方？同一个故障是否

有更加严重的问题?

3. 验证和分析

有些错误可能是回归期间发现的,这主要是由于二次修改造成的,即把原来没有问题的功能改错了。在对这种缺陷进行报告时应特别标识,这样可以方便测试组成员及早发现和测试与之有关联的功能。另外,对缺陷进行适度的分析可以帮助开发人员更快地进行错误定位。对缺陷严重度的分析,可以提升缺陷的修改优先级,使重要的问题优先得到修改。

4. 精简和歧义

缺陷报告要多阅读几遍,尽量使用短语叙述,不使用地方俚语,使阅读理解的时间尽量缩短,这是“雅”的一部分。

另外,测试组提交缺陷报告的一个好的做法是成员交叉评审,即测试组长将缺陷报告汇集在一起,打乱顺序后分发给各个组员,尝试复现和确认,这种角色转换式的评审对于剔除一些不是很有价值的缺陷是很有帮助的。

11.3.4 缺陷处理流程及状态跟踪

软件缺陷是软件开发过程中的“副产品”。缺陷会存在于软件产品的整个生命周期中:可以是软件代码的问题、系统文档(开发文档和测试文档等)存在问题,或者是用户的帮助文档和使用指南方面的问题等。

测试是发现缺陷的主要手段,也是它的主要目的。测试活动和开发活动一样,是项目质量保证不可或缺的重要部分。因此,对于测试活动的主要产物:缺陷,我们需要建立一个完善的缺陷管理流程,来对缺陷进行报告、查询、分类、跟踪、处理和验证等。

缺陷状态及处理流程没有一个统一的标准,一般都是企业根据自身的组织结构和业务流程进行自我定制。图 11-4 所示的是一个缺陷处理流程的一个范例,本书以此为例介绍一下缺陷处理的过程。

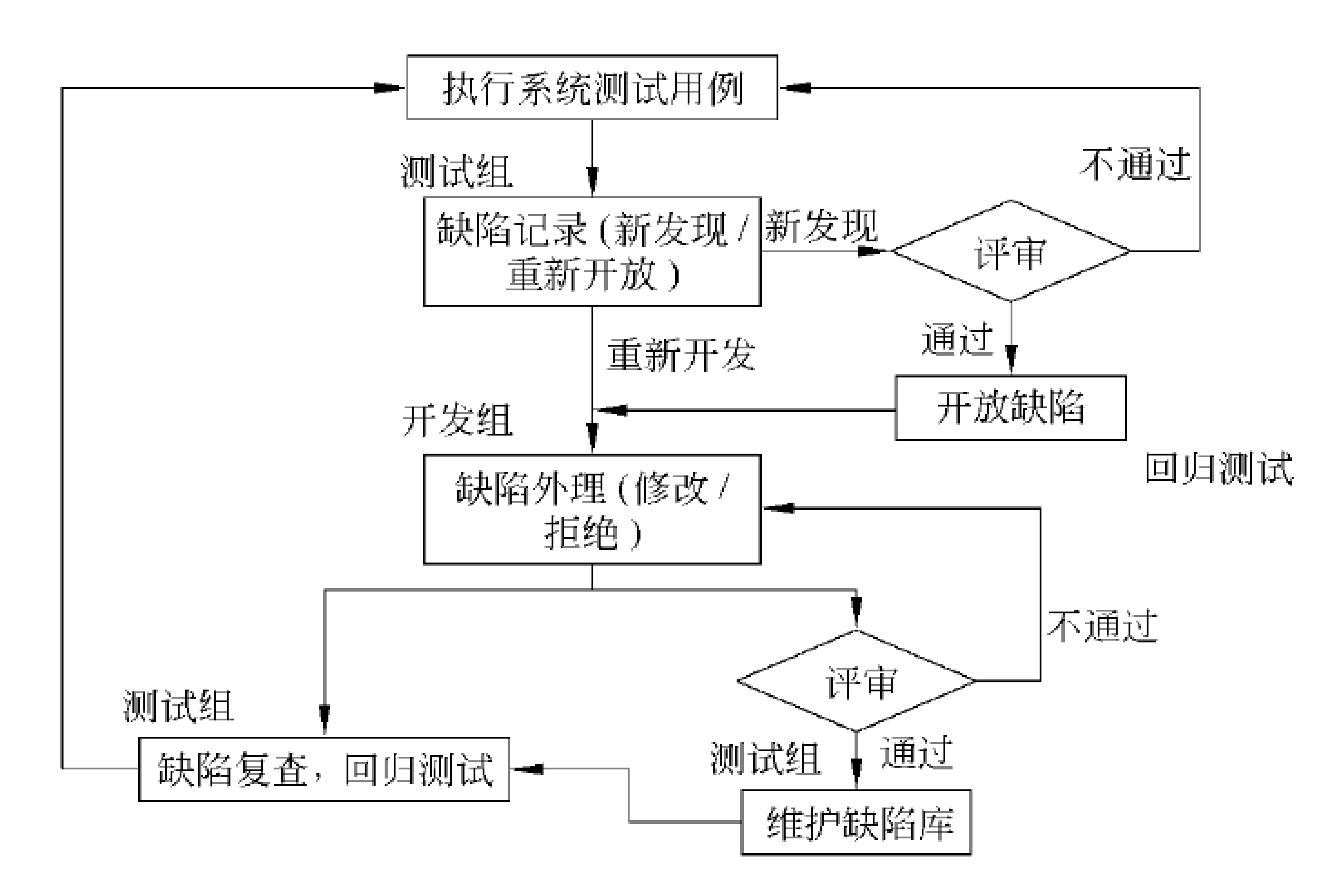


图 11-4 缺陷处理流程

在缺陷分类一节已经对缺陷状态这个属性进行了说明，缺陷状态指缺陷在一个跟踪修复过程的状态迁移情况，包括 New(提交)、Open(开放)、Reopen(重新开放)、Fixed(已修复)、Closed(关闭)及 Rejected(拒绝修复)等，具体介绍见表 11-2。

表 11-2　状态迁移情况

状态标识	表意	描　述	责任人
New	新发现	提交新缺陷的标识状态	测试人员
Open	开放	缺陷确认后的状态，前一个状态是 New	开发组长/项目经理
Reopen	重新开放	对已修改缺陷进行验证后没有通过所标识的状态；或者已经关闭的缺陷，在回归测试期间又出现错误。一般 fixed 变更到 Reopen，Closed 变更到 Reopen 或者 Rejected 变更到 Reopen	测试人员
Fixed	已修复	缺陷修复后测试前所标识的状态	开发人员
Closed	关闭	已修改缺陷验证后通过所标识的状态	测试人员
Rejected	拒绝	确认不是缺陷的缺陷或者因缺陷报告描述不清、重复、不能复现、不采纳所提意见建议、或虽然是个错误但还没到非改不可的地步故可忽略不计、或者测试人员误报，从而被拒绝的缺陷	开发组长/项目经理/开发人员

根据缺陷状态的这个定义，缺陷处理流程可以做如下描述：

测试人员执行测试用例后，报告所发现的缺陷(New)。开发组长(或者项目经理)对缺陷进行确认，确认后的缺陷状态置为 Open，并分发给项目组开发人员。开发人员对缺陷进行确认并回复，是拒绝(Rejected)还是接受，拒绝要给出理由；接受则着手对缺陷进行修改，并将得到修复的缺陷状态置为 Fixed，并交由测试人员验证，也就是回归测试，确保缺陷被修复且没有引入新的缺陷。

在 CMMI 中，其实一个确认的缺陷，就是一个变更，由变更控制委员会来分析决定这个缺陷是拒绝还是修改，若要修改就对此缺陷进行部署分配及改完日期，然后开发人员来修改，测试人员来复核，最后也是由变更控制委员会决定这个变更的结束。

以上缺陷处理的过程也许是个简单的过程，但是缺陷状态的频繁变更管理起来却是困难的，另外我们也需要对测试中发现的问题进行统计分析，比如各类缺陷的出现数量、缺陷在测试周期内的变化趋势等，以揭示更多的质量信息。所以，现在大多数的软件企业都构建了自动化测试管理平台，并在缺陷变更控制的基础上扩充了测试需求管理、测试用例设计等功能以适应软件测试发展的要求，如本书中介绍使用的 TestDirector 就是其中的佼佼者(有关 TestDirector 请参考第 18 章有关内容)。

我们前面在讲测试执行过程监控期间，曾经给出了一些指标公式，现在根据本节对缺陷管理知识的进一步了解，可以给出更为准确的表达式(见表 11-3)。

表 11-3 指标表达式

指标描述	指 标 公 式
缺陷有效率	Σ(Opened,Fixed,Reopened,Closed) /Σ(Opened,Fixed,Reopened,Rejected,New,Closed) * 100%
缺陷确认率	Σ(Opened,Fixed,Reopened,Rejected,Closed) /Σ(Opened,Fixed,Reopened,Rejected,New,Closed) * 100%
缺陷修改率	Σ(Fixed,Closed)/Σ(Opened,Fixed,Reopened,Rejected,Closed) * 100%
缺陷验证率	Σ(Reopened,Closed)/Σ(Reopened,Closed,Fixed) * 100%

11.4 回归测试

为了确保软件在完善后、缺陷修复后或者任何其他变更不会导致原有功能的失效,通常需要重新对软件发生改变的部分进行测试,这个过程称为回归测试。

当前,迭代开发广泛存在于大多数软件开发流程中,每一次迭代都会带来软件某种程度上的改变,我们需要对这些改变作出确认,因此回归测试尤其显得重要。

系统任何时候发生变更时,都应及时开展回归测试,这种测试可以是系统全范围的,也可以选择性进行,例如在关联性分析结果指导下选择性的只针对发生变更的局部进行回归测试。但是,通常在一轮测试结束后或者是几轮局部的回归测试之后我们应该执行一次全范围的回归测试。在回归测试期间,如何选取充足的恰当的测试用例获取最大的收益(比如节约时间、充分的测试覆盖等)是很重要的。一般回归测试的用例基于以下两点考虑:一是系统中哪些功能得到了完善,哪些缺陷已经修改,哪些方面相对于前一个实现发生了变更?二是系统的这些改变在哪些方面对系统有影响。

回归测试专注于系统的变更所带来的影响,在大多数企业或者组织,一般优先进行缺陷的回归。在测试完成标准中,通常会将回归测试零缺陷作为一个判定指标。

11.4.1 回归测试方法

在软件生命周期中,即使一个得到良好维护的测试用例库也可能变得相当大,这使每次回归测试都重新运行完整的测试用例变得不切实际。因此,选择回归测试方法应该兼顾效率和有效性两个方面,根据项目实际情况,达到平衡。常用的选择回归测试的方式包括以下几种。

1. 再测试全部用例

选择测试用例库中的全部测试用例组成回归测试包,这是比较安全的方法,具有最低的遗漏回归错误的风险。再测试全部用例几乎可以应用到任何情况下,基本上不需要进行用例分析和设计,但它测试成本很高,随着开发工作的进展,测试用例不断增多,重复原先所有的测试将带来很大的工作量,往往超出了我们的预算和进度。

2．基于风险选择测试

基于一定的风险标准从测试用例库中选择回归测试包。首先运行最重要的、关键的和可疑的测试，跳过那些次要的、优先级别低的测试用例或那些功能相对很稳定的模块。执行那些次要用例时，即便发现缺陷，这些缺陷的严重性也较低。

3．基于操作剖面选择测试

如果测试用例是基于软件操作剖面开发的，测试用例的分布情况反映了系统的实际使用情况。回归测试所使用的测试用例个数可以由测试预算确定，回归测试可以优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，有助于尽早发现那些对可靠性有最大影响的故障。这种方法可以在一个给定的预算下最有效的提高系统可靠性，但实施起来有一定的难度。

4．再测试修改的部分

当测试者对修改的局部化有足够的信心时，可以通过相依性分析识别软件的修改情况并分析修改的影响，将回归测试局限于被改变的模块和它的接口上。通常，一个回归错误一定涉及一个新的、修改的或删除的代码段。在允许的条件下，回归测试尽可能覆盖受到影响的部分。这种方法可以在一个给定的预算下有效地提高系统可靠性，但需要良好的经验和深入的代码分析。

11.4.2　回归测试过程

回归测试的实施过程按顺序可以分为以下几步：

（1）测试分析员根据测试问题报告和软件变更报告单，分析回归测试的测试范围，并确定原软件测试的充分性要求、终止要求、资源要求、软件特性、测试技术和方法的适用程度，并酌情变更，确定回归测试的测试进度，完成软件回归测试计划。

（2）测试设计员和测试程序员根据软件回归测试计划确定测试用例，可从原软件测试说明中选择测试用例、或修改原有测试用例、或设计新的测试用例，补充相应的测试数据、测试资源和测试软件，建立相应的测试环境，确定相应的测试顺序，编写软件回归测试说明。

（3）测试员和测试分析员按照软件回归测试说明对变更的软件进行测试。

（4）测试分析员根据原测试问题报告、原软件变更报告单，软件回归测试计划、测试说明、测试记录、软件问题报告单对回归测试的工作进行总结，编写软件回归测试报告、测试问题报告。

回归测试完成后形成的文档一般应有：回归测试计划、回归测试说明、回归测试报告、回归测试记录、回归测试问题报告，其编写格式与初次测试没有什么区别。

11.5　HRMIS 的测试执行过程

在第 9 章介绍测试设计的内容时，我们以 HRMIS 为例从单元测试、集成测试和系统测试角度对 HRMIS 的测试需求进行了分析整理，并有针对性地选择设计了部分测试用

例，在实际工作中，除了手工设计用例，执行测试之外，有时为了提高工作效率或者满足特定的测试需求等，还需要进行一些自动化测试，此时往往需要自行编写一些测试脚本。

11.5.1 测试环境搭建

1. 设备选型和构建

本例中并没有涉及特殊设备，因此在选择测试设备时自由度相对较大，但是通过前面的内容大家已经了解到 HRMIS 的系统测试还包含一部分效率的考查，也就是说要做一部分性能的测试，因此我们在选择设备时应对 HRMIS 的目标生产环境进行调研，使测试环境能够与其生产环境匹配，从而使效率测试结果能够更具参考价值。

根据 HRMIS 的 C/S 系统结构，我们选择一台 PC 服务器作为数据库服务器，2 台 PC 作为客户机，其中一台安装配置共享打印机，以备报表打印测试(如图 11-5 所示)。这些设备构成被测系统的基础硬件设施，详细清单见表 11-4 和表 11-5。

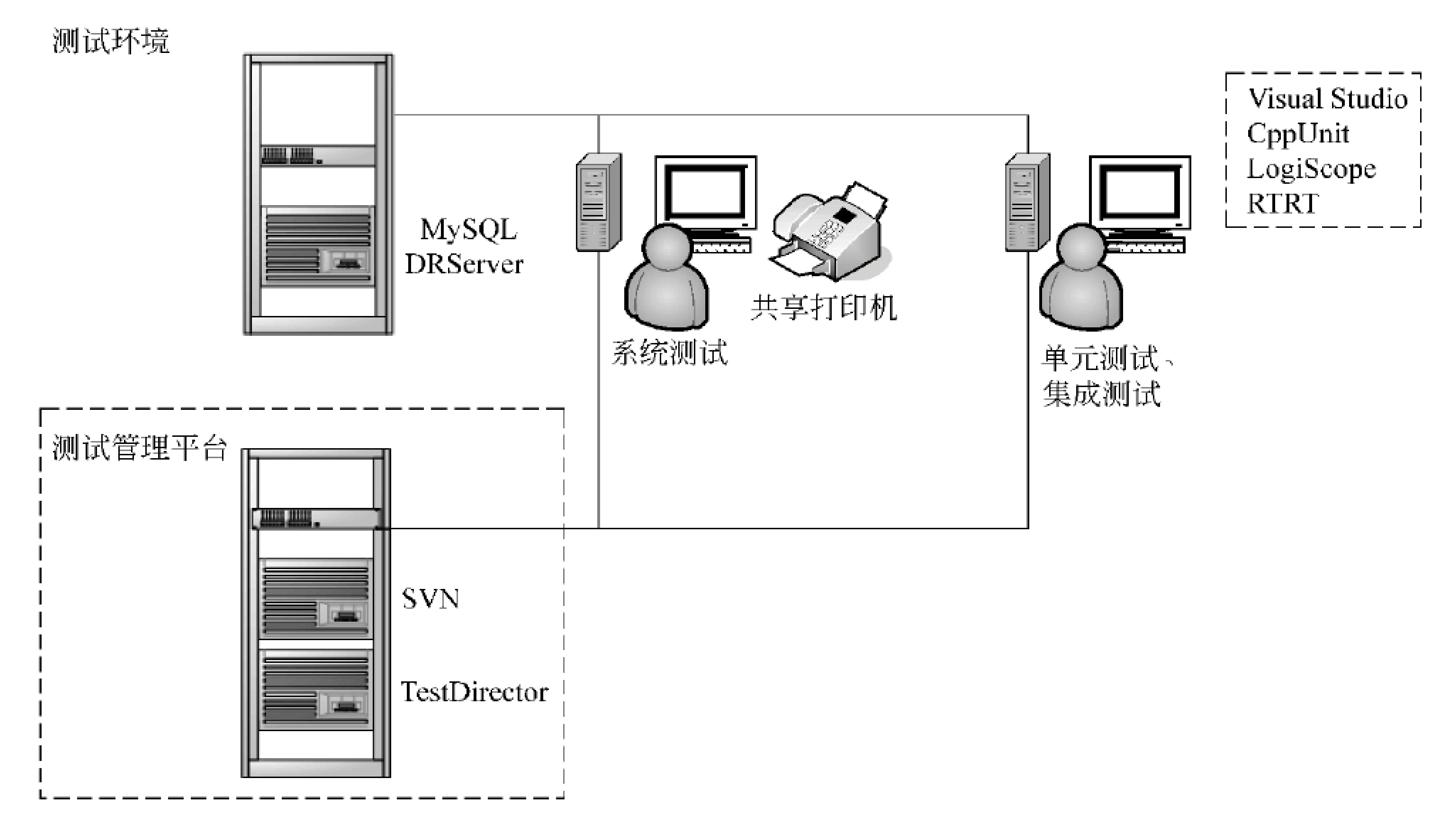


图 11-5 HRMIS 的测试环境

表 11-4 数据库服务器硬件环境

CPU 种类及数量	Intel Xeon CPU，2.0GHz，2 个
内存	2GB
硬盘	160GB
网卡	RTL8139 10/100 Mbps Ethernet PCI Adapter

表 11-5　客户端硬件环境

CPU 种类及数量	Intel Pentium 4，1.1GHz，1 个
内存	512MB
硬盘	80GB
网卡	100MB

2. 软件安装要求

HRMIS 适用于中小型企业，没有跨平台要求，因此我们只考虑 Windows 系列一种操作系统即可(本例中我们选用的是 Windows XP，如果必要还应测试其他操作系统，如 Windows 2000、Windows 2003 或者 Vista)。为了防止病毒对测试环境的影响，可以选用正版卡巴斯基 6.0 作为杀毒软件，同时保持测试环境的网络独立于其他办公网络。另外，在本例中我们对 HRMIS 的部分单元要进行白盒测试，因此我们要预先安装配置所选用的白盒测试工具。综合以上分析，我们可以得出支持软件的安装清单见表 11-6～表 11-9。

表 11-6　数据库服务器

操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
数据库管理系统	MySQL 6.0

表 11-7　客户端 1 软件环境

操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
MySQL 数据库访问接口	MyODBC 3.5.1
开发工具	Visual Studio 6.0
白盒测试工具	CppUnit1.12.0
	Logiscope 6.1
	Rational TestRealtime

表 11-8　客户端 2 软件环境

操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
MySQL 数据库访问接口	MyODBC 3.5.1

表 11-9　被测软件系统

软件名称	人力资源管理系统(HRMIS)
软件版本	0.2
说明	需要在两台客户机安装

最后，重新检查确认所准备的测试环境与所列出的硬件、软件是否一致，确认无误后我们将目标测试系统植入到测试环境中。

3. 测试管理平台

测试过程需要有条不紊的进行，测试的产出物需要得到有效的管理和控制。因此，除了构建测试环境之外，一个完整规范的测试环境应该还包括测试管理环境的配置（如图 11-5 所示虚线框区域）。

本案另外选型两台 PC 服务器，用以安装配置我们的管理平台。

1）配置管理系统

Subversion（简称 SVN）是近年来崛起的版本管理工具，号称是 CVS 的替代品，SVN 以其灵活性、易用性和免费获得了越来越多的项目组织的青睐。在本案中，我们同样使用 SVN 作为配置管理系统，并设置如下配置项：售前文件、测试计划、测试用例、会议纪要、阶段报告和其他过程文件（有关 SVN 的使用不在本书的介绍范围之内，如感兴趣可参考有关资料）

2）测试管理系统

本案中选用 TestDirector 8.0 作为我们的测试管理系统，完成测试需求的跟踪、测试用例的设计和执行、缺陷的录入和处理跟踪。

11.5.2　单元测试执行情况

1. 应用 CppUnit 的测试执行

第 10 章介绍测试方法和用例设计时，针对 CMClerk 单元测试设计了三个测试用例，让我们再来回顾一下。

【用例 11-1】　语句覆盖和路径覆盖

用例名称	测试 AddClerk	用例标识	P200901UT-YGGL-YGLR-01
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01		
用例说明	测试员工信息管理类保存员工信息（AddClerk）方法，对于给定的合法数据能否正常保存		

续表

用例的初始化	硬件配置	无特殊要求			
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库			
	测试配置	无特殊要求			
	参数设置	部门组织结构已初始化			
操作过程					
序号	输入及操作说明		期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据（包括姓名、性别、年龄、出生日期、身份证号（身份证号要满足前置条件）等）		—	—	
2	将桩模块数据传递到 AddClerk 中		—	—	
3	调用方法 AddClerk		—	—	
4	查询数据库，检验数据保存是否正确		员工信息成功保存	可以查询到新录入的员工信息	
5	查看函数返回值是否为 1				
前提和约束			员工身份证号在已保存的员工信息中不存在		
过程终止条件			无		
结果评价标准			身份证号为 370205198612301234 的员工信息保存成功，函数返回 1		
设计人员			—	设计日期	—

桩数据如下：

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "370205198612301234"
CString strYGSex="男";
COleVariant oleBirth("1986-12-30");
COleDateTime dtBirthDay=oleBirth;
CString strYGPolity="民主联盟";
CString strYGCulture="硕士";
CString strYGMarry="已婚";
CString strYGHome="青岛市环海路 465 号";
CString strYGPhone="86261283";
CString strYGMobile="18978273909";
CString strCallWho="林轩";
CString strCallNo="13628390987";
CString strYGGradu="中国农业大学信息工程学院";
```

```
COleVariant oleInTime("1986-12-30");
COleDateTime dtInTime=oleInTime;
CString strYGRole="评测工程师";
CString strYGState="试用";
CString strYGNation="哈萨克族";
CString strYGDepart="1";                //与合同的测试用例要一致
CString strYGDuty="负责应用软件的测试、缺陷记录及报告编写";
```

【用例 11-2】 语句覆盖和路径覆盖

<table>
<tr><td>用例名称</td><td colspan="2">测试 AddClerk</td><td>用例标识</td><td colspan="2">P200901UT-YGGL-YGLR-02</td></tr>
<tr><td>测试追踪</td><td colspan="5">业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01</td></tr>
<tr><td>用例说明</td><td colspan="5">测试员工信息管理类保存员工信息(AddClerk)方法，数据重复时保存失败</td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="4">无特殊要求</td></tr>
<tr><td>软件配置</td><td colspan="4">软件可以正常启动运行，正常连接 MySQL 数据库</td></tr>
<tr><td>测试配置</td><td colspan="4">无特殊要求</td></tr>
<tr><td>参数设置</td><td colspan="4">部门组织结构已初始化</td></tr>
<tr><td colspan="6">操作过程</td></tr>
<tr><td>序号</td><td colspan="2">输入及操作说明</td><td>期望的测试结果</td><td>评价标准</td><td>备　注</td></tr>
<tr><td>1</td><td colspan="2">在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号(身份证号要满足前置条件)等)</td><td>—</td><td>—</td><td>桩数据同用例 P200901UT-YGGL-YGLR-01</td></tr>
<tr><td>2</td><td colspan="2">将桩模块数据传递到 AddClerk 中</td><td>—</td><td>—</td><td></td></tr>
<tr><td>3</td><td colspan="2">调用方法 AddClerk</td><td>—</td><td>数据保存失败</td><td></td></tr>
<tr><td>4</td><td colspan="2">查看函数返回值是否为－1</td><td></td><td></td><td></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td></td><td></td></tr>
<tr><td colspan="3">前提和约束</td><td colspan="3">• 员工身份证号在已保存的员工信息中已存在
• P200901UT-YGGL-YGLR-01 通过</td></tr>
<tr><td colspan="3">过程终止条件</td><td colspan="3">无</td></tr>
<tr><td colspan="3">结果评价标准</td><td colspan="3">身份证号为 370205198612301234 的员工信息保存失败，函数返回－1</td></tr>
<tr><td colspan="3">设计人员</td><td>—</td><td>设计日期</td><td>—</td></tr>
</table>

【用例 11-3】 边界值测试

用例名称	测试 AddClerk	用例标识	P200901UT-YGGL-YGLR-03	
测试追踪	业务需求说明书:人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中,TR01			
用例说明	测试员工信息管理类保存员工信息(AddClerk)方法,测试含有字母的身份证号码的处理情况			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行,正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	在桩模块中构造一组员工信息的测试数据(包括姓名、性别、年龄、出生日期、身份证号(身份证号要满足前置条件且末尾为字母)等)	—	—	参见桩数据
2	将桩模块数据传递到 AddClerk 中	—	—	
3	调用方法 AddClerk	—	—	
4	查询数据库,检验数据保存是否正确	员工信息成功保存	可以查询到新录入的员工信息	
5	查看函数返回值是否为 1		函数返回 1	
前提和约束		—		
过程终止条件		无		
结果评价标准		身份证号为 37020519861230123X 的员工信息保存成功,函数返回 1		
设计人员		—	设计日期	—

桩数据如下:

```
#define strYGName "艾玛·布拉提斯"
#define strYGPriCode "37020519861230123X"
CString strYGSex="男";
COleVariant oleBirth("1986-12-30");
COleDateTime dtBirthDay=oleBirth;
CString strYGPolity="民主联盟";
CString strYGCulture="硕士";
CString strYGMarry="已婚";
CString strYGHome="青岛市环海路 465 号";
```

```
CString strYGPhone="86261283";
CString strYGMobile="18978273909";
CString strCallWho="林轩";
CString strCallNo="13628390987";
CString strYGGradu="中国农业大学信息工程学院";
COleVariant oleInTime("1986-12-30");
COleDateTime dtInTime=oleInTime;
CString strYGRole="评测工程师";
CString strYGState="试用";
CString strYGNation="哈萨克族";
CString strYGDepart="1";                    //与合同的测试用例要一致
CString strYGDuty="负责应用软件的测试、缺陷记录及报告编写";
```

用例 11-1 和用例 11-2 设计的目的是完成 CMClerk 的 AddClerk 的语句覆盖和两条基本路径的覆盖，用例 11-3 是根据边界的分析补充的一组测试用例。我们已经知道，单元测试如果在一个类的级别需要准备驱动模块和桩模块才能完成动态测试，我们前面介绍的 CppUnit 这个单元测试框架一定程度上扮演了驱动模块的作用，使我们可以不用关注驱动模块的设计，而专注于测试用例本身的设计和编写。实际上，对于用例 11-1 和用例 11-2，在介绍应用 CppUnit 编写单元测试用例时，已经把这两个用例以测试代码的形式予以实现。下面来看一下基于 CppUnit 的单元测试用例执行过程。

通过调用 CppUnit::MfcUi::TestRunner 引入 CppUnit 的可视化测试执行界面（如下调用代码，前已讲述），这样当运行程序时会自动调用 CppUnit 的测试执行程序（如图 11-6（a）所示）。

```
CppUnit::MfcUi::TestRunner runner;
runner.addTest( CppUnit::TestFactoryRegistry::getRegistry().makeTest() );
```

TestRunner 从上至下依次为测试用例选择栏，单击右边的 Browse 可以选择测试用例（如图 11-6 所示），即我们在测试工厂中注册的测试用例，本例中是 CMClerkTestCase：AddClerk、CMClerkTestCase：IsExist、CMClerkTestCase：SetClerkSex、CMClerkTestCase：SetClerkState、CMClerkTestCase：RemoveClerk 等 5 个测试用例。

Progress 显示测试执行进度及测试执行情况，按照三种状态统计，分别是 Runs、Error 和 Failure。所有执行过程中出现的 Error 和 Failure 都将列示在下面的信息栏中，选择相应的信息条目则在 Detail 中给出具体的错误信息。

TestRunner 右边的按钮基本上可以望文生义，操作非常简单。

- Run：执行测试选中的用例。
- Stop：停止执行。
- Close：关闭 TestRunner。

图 11-7 和图 11-8 所示的分别是在测试用例执行出错和执行成功的情况下的显示状态。

图 11-7 为执行了 12 个测试用例，有一个出错信息。



(a) Test Runner界面



(b) Test hierrarchy 界面

图 11-6　TestRunner 界面布局

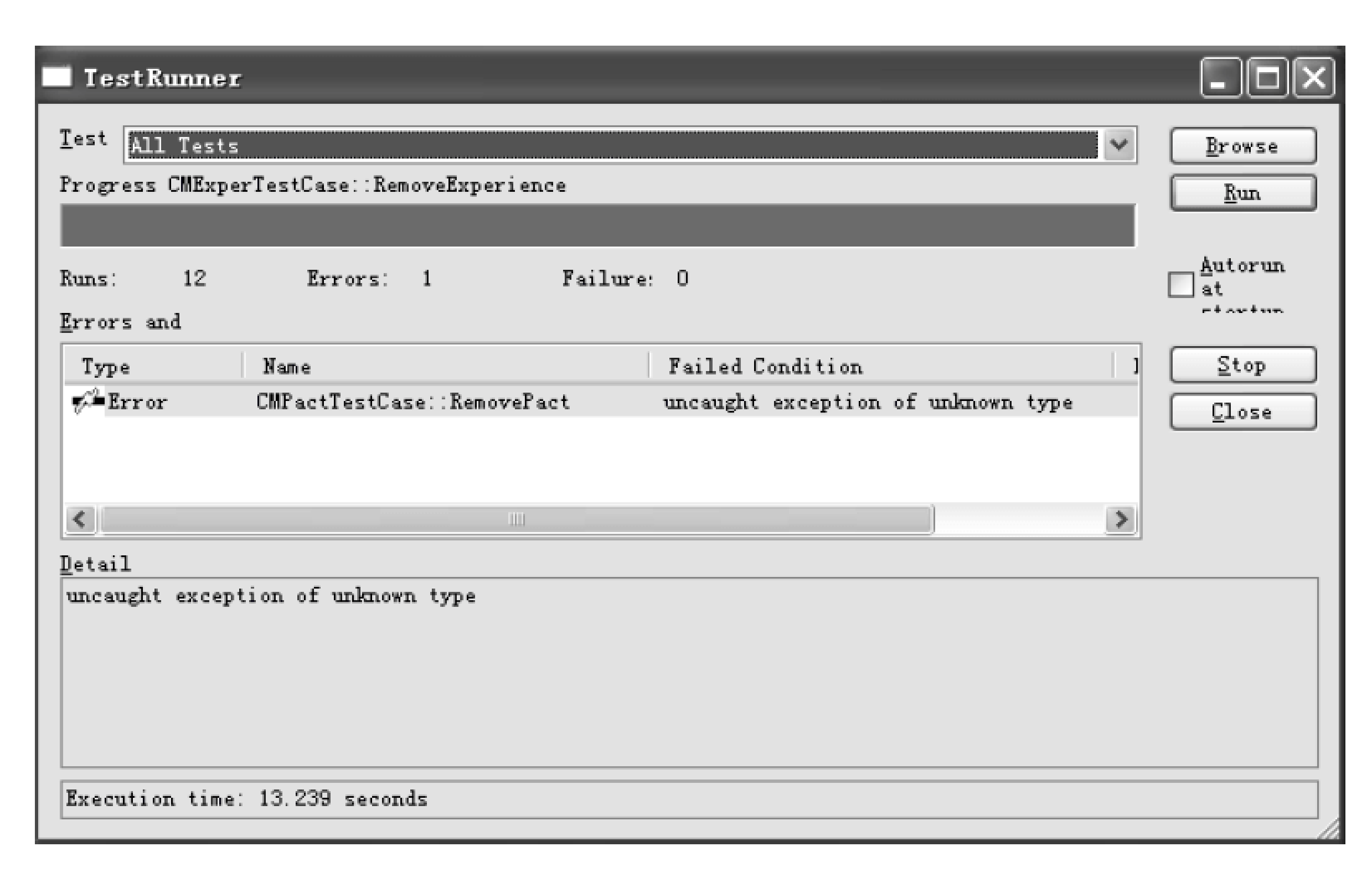


图 11-7　TestRunner 执行出错界面

图 11-8 为执行 1 个用例，没有出错和失败信息。

我们在本例中主要使用了 CPPUNIT_ASSERT(condition) 和 CPPUNIT_ASSERT_EQUAL(expected,current) 这两个宏，事实上 CppUnit 测试框架提供了相当丰富的测试断言宏，下面逐一解释。

- CPPUNIT_FAIL(message)：表示失败，message 中用于存储和输出诊断信息。
- CPPUNIT_ASSERT_MESSAGE(message, condition)：断言条件 condition 为真，若为假，message 中指明了诊断信息。
- CPPUNIT_ASSERT_EQUAL_MESSAGE(message,expected,current)：断言两个值相等，message 中指明了附加的诊断信息。
- CPPUNIT_ASSERT_DOUBLES_EQUAL(expected,current,delta)：断言两个值不精确相等。

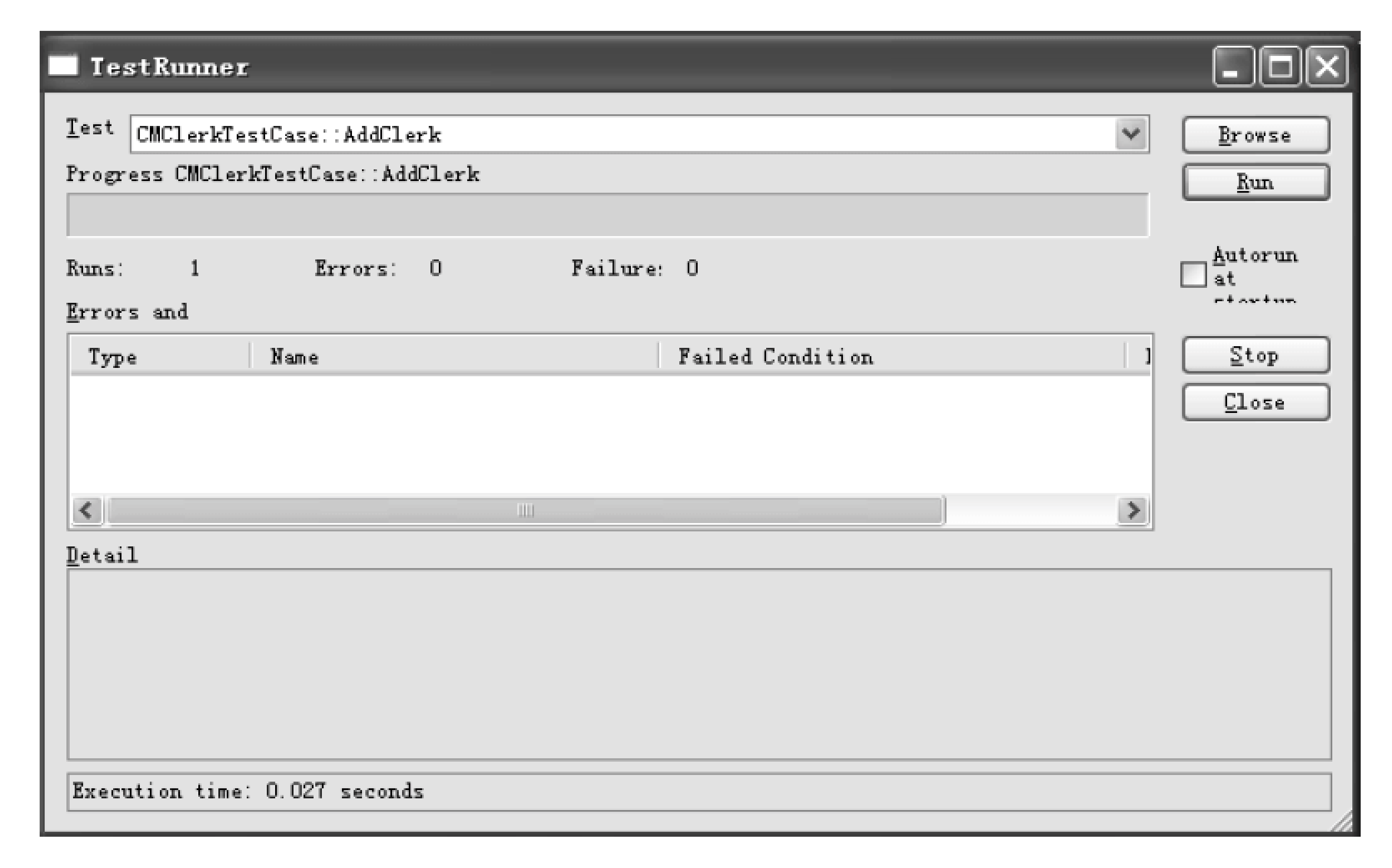


图 11-8 TestRunner 执行成功界面

关于 CPPUNIT_ASSERT_EQUAL 有一点需要说明，该宏对于 expected 和 actual 是有要求的：

- 具有相同的类型（比如都是 std∷string）。
- 可以使用＜＜序列化到 std∷strstream（assertion_traits＜T＞∷toString 中指明）。
- 能用＝＝作比较（assertion_traits＜T＞∷equal 中指明）。

测试断言本质上可以说是预期结果和实际结果的自动化比较和判断。

下面介绍案例注意事项。

本例中，TestCase 的函数体内需要访问数据库，故调用 TestRunner 之前要先建立数据库连接，否则在 TestRunner 执行结果中会出现“uncaught exception of unknown type”的错误信息，这种错误信息不明确，且不能定位分析，如图 11-9 所示。

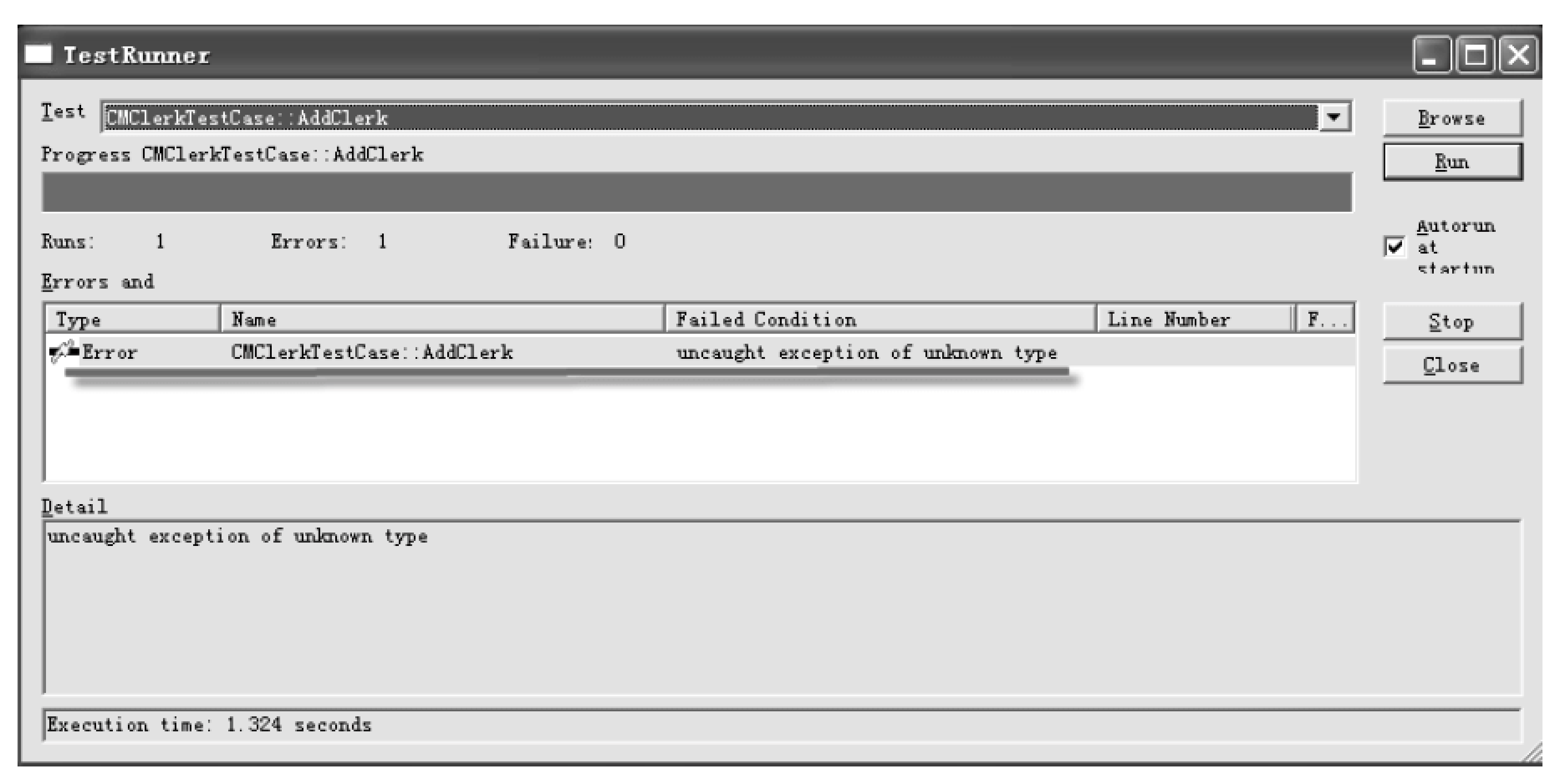


图 11-9 TestRunner 执行错误信息

关于 CMClerk 的动态单元测试就讨论到这里，在本项目的实际的单元测试过程中我们还使用了一些其他的测试手段，比如人工代码走查，在进行代码走查的过程中我们也发现了该单元的其他一些错误，作为实例，列举如下：

（1）数组索引使用不当。

```
void CMDepart::GetDescripe(int cDepartID, CString &sDescripe)
{
    int nCont=a_DepartID.GetUpperBound();
    int iDx=0;

    //TestEx:边界值测试
    //2008-12-14
    //示例错误代码
    //while (iDx<nCont) {

    while (iDx<=nCont) {
        if (atoi(a_DepartID.GetAt(iDx))==cDepartID) {
            sDescripe=a_Descripe.GetAt(iDx);
            return;
        }

        iDx++;
    }
}
```

【点评】

使用 CStringArray 的 GeUpperBound 方法取得的数组上标是数组中最后一个元素的标号，所以此处循环处理条件应该是 while（iDx＜＝nCont）而不是 while（iDx＜nCont）。

（2）“死”代码。

```
//判断指定的部门下是否有员工
BOOL CMDepart::HaveClerk(CString cDepartID)
{
    CString strSql;
    strSql.Format("SELECT * FROM T_YG WHERE BMBH='%s'",cDepartID);

    CADORecordset* pRS=new CADORecordset(theApp.pHr_AdoDb);
    pRS->Open((LPCTSTR)strSql);

    if (pRS->IsEOF())
```

```
    {
        //TestEx: 正确的做法
        pRS->Close();
        delete pRS;

        return false;
    }
    else
    {
        //TestEx: 正确的做法
        pRS->Close();
        delete pRS;

        return true;
    }

/*
TestEx: 代码检查实例——死代码

    pRS->Close();
    delete pRS;
*/
}
```

【点评】

代码中黄色标注区是原始代码，因为该函数需要返回 bool 值，黄色标注的代码在 return 指令后是根本不可能执行的，这就是我们所说的“死”代码。这么做的结果就是造成内存泄露，分配给 pRS 的内存始终得不到释放。

2. 应用 LogiScope 的静态分析

Telelogic LogiScope 是 IBM 公司研制的白盒测试工具，支持 C、C++ 、Java 等主流编程语言，应用 LogiScope 可以完成代码规则审查、调用关系分析等。前一节简单介绍了代码的人工审查，其实代码审查工作也可以借助 LogiScope 进行，这样能够获得更高的效率和准确率。在本案 HRMIS 的单元测试过程中，也部分应用了 LogiScope 进行静态分析，以获得单元模块的更全面的测试结果。

1）代码规则审查

代码规则审查主要依据 LogiScope 自带的编码规则集进行，主要按如下步骤进行：

① 确定测试内容和编码规则集。

② 确定本次测试的测试方法。

③ 执行测试。

④ 测试结果分析汇总。

在对本案 HRMIS 的单元测试中，主要应用 LogiScope 的 C++ 编码规则，具体测试步骤如下：

① 建立代码审查工程文件，选择被测代码所在的目录。

② 选择编译器以及测试用规则集，LogiScope 中可供选择的编译器有：

- C with Microsoft Developer Studio 6.0；
- C with Microsoft Developer Studio 5.0；
- C with Microsoft Developer Studio 4.x；
- Microsoft C 1.5；
- ANSI C；
- GNU C；
- MICROTEC Research C ANSI mode；
- DIAB C；
- SUN C。

测试规则集的选择，可以选择系统默认的编程规则集，也可以选择用户自定义编程规则集，编程规则集根据语言可分为 C、C++ 规则集。用户可以根据代码审查的需要选择全部或者部分的规则集进行测试。

通过以上步骤建立起代码审查工程，工程建立后首先对程序进行编译，然后通过自动生成的报告即可看到代码审查的结果，审查结果包括三部分内容：

- Violated Rules：列出了违反相关规则的代码，表明具体违反什么规则、定位问题代码所在的函数、行数。
- Clean Rules：列出了本次测试程序符合的具体编码规则。
- Ignored Rules：列出了本次测试未使用的规则。

在对 HRMIS 进行代码审查之后，得到如表 11-10 和表 11-11 所示的审查结果。

表 11-10　违反的规则

<table>
<tr><th>序号</th><th>规则名</th><th>说　　明</th><th>违反文件(文件名、行号)</th></tr>
<tr><td>1</td><td>ansi</td><td>函数的声明和实现必须符合 ANSI 语法
(1) 函数的所有参数必须命名，并且必须在函数声明中说明各个参数的类型
(2) 禁止函数的参数为空</td><td>Ado2. cpp：85、90、116、129、141、144、147
Oledb2. cpp:31、36
HeaderToolbar. cpp：17、44、267、323、342、354
AdvComboEdit. cpp:22、27、43</td></tr>
<tr><td>2</td><td>asscon</td><td>不允许在 if，while，for，switch 控制指令中的条件表达式中使用赋值运算符(=，+=，-=，*=，/=，%=，>>=，<<=，&=，|=，^=，++，--)</td><td>Ado2. cpp:1157、1164
Oledb2. cpp:47
TreeCtrlEx. cpp:82、85</td></tr>
</table>

续表

序号	规则名	说　明	违反文件(文件名、行号)
3	brkcont	Break、continue 指令禁止使用在控制语句(for，do，while)的条件表达式中。然而，break 指令被允许使用在 switch 语句的结构体中	XListCtrl. cpp：873、2385、3293、3299、3318、3324 ClerkChange：108 DlgAllTrain：81 DlgWork. cpp：128
4	ctrlblock	在控制语句段(if，for，while，do)中必须使用{}	Ado2. cpp：93、101、111、114、121、153、160 XCombo. cpp：43 XEdit. cpp：54、64、76、136、159 XHeaderCtrl. cpp：87、94、164、230、244、285
5	delarray	当删除数组时必须使用[]	XThemeHelper. cpp：232、344 MClerk. cpp：97、203、222、264、270 MDepart. cpp：89、108、124、132、177、198 UserAdmin. cpp：166

表 11-11　遵守的规则

序号	规则名	说　明
1	assignthis	在赋值操作符的定义中，必须检查两个参数(this 或者 * this)是否相等，如果相等，必须返回 * this
2	Convnewdel	重写 new 和 delete 操作符时，必须保持和系统默认的 new、delete 的行为一致。new 必须遵守：返回值必须是 void *，第一个参数类型必须是 site_t，delete 必须遵守：返回值必须是 void，第一个参数必须是 void *，第二个参数类型必须是 site_t
3	delifnew	如果在一个类中声明了操作符 new 就要在该类中同时声明操作符 delete
4	frndclass	如果使用友元类，则友元类必须在类的开始处被声明(在类的成员被声明之前)
5	goto	goto 语句不能使用，某些特定的标签(label)可以使用 goto
6	inlinevirt	虚函数不能被声明为 inline
7	normalnew	如果在一个类中定义了一次或者多次 new 操作符，则这些定义中，必须有一个遵守 new 的一般模式：第一个参数的类型必须是 size_t，如果还有其他参数，每个参数都必须有一个默认值
8	nounion	关键字 union 不允许使用
9	overload	“&&”、“‖”和“,”三个操作符不允许重载

通过这种代码规则审查，可以很容易地发现源代码中编写不规范的地方，比如查看上述列表中的第 3 条，我们可以看到 break 语句被禁止在条件控制语句中使用，检查结果中显示 ClerkChange. cpp 中第 108 行违规，打开 ClerkChange. cpp 文件找到对应行就可以

看到 break 语句被用在了 if 条件控制块中(如图 11-10 所示)。

```
// add columns
for (i = 0; ; i++)
{
    if (lpszHeaders[i] == NULL)
        break;

    lvcolumn.mask      = LVCF_FMT | LVCF_SUBITEM | LVCF_TEXT | LVCF_WIDTH | LVCF_IMAGE;
    lvcolumn.iSubItem = i;
    lvcolumn.pszText  = lpszHeaders[i];
    lvcolumn.cx       = (lpszHeaders[i+1] == NULL) ? w - total_cx - 2 : (w * m_nChange_ColWidths[i]) / 58;

    total_cx += lvcolumn.cx;
    pList->InsertColumn(i, &lvcolumn);
}
```

图 11-10　ClerkChange. cpp 源码

再如,查看第 2 条违规信息,规则是“不允许在 if , while , for , switch 控制指令中的条件表达式中使用赋值运算符(=, +=, -=, *=, /=, %=, >>=, <<=, &=, |=, ^=,++,--)”,检查结果中显示,TreeCtrlEx. cpp 文件的第 82 行违反规则,我们打开这个文件找到对应的行,即可以看到这个违规代码(如图 11-11 所示)。

```
HTREEITEM CTreeCtrlEx::FindItemData(HTREEITEM hti,DWORD dwData)
{
    if(hti == NULL) return NULL;
    if(GetItemData( hti ) == dwData)
        {
        Select( hti, TVGN_CARET);
        EnsureVisible( hti );
                return hti;
        }

    hti = GetChildItem( hti );
    do
    {
        HTREEITEM hti_res;
        if((hti_res = FindItemData( hti, dwData)) !=NULL )
                return hti_res;

    }while( (hti = GetNextSiblingItem( hti )) != NULL );
    return NULL;
}
```

图 11-11　TreeCtrlEx. cpp 源码

这是一种不规范的代码写法,可读性差,好的写法应该是这样的:

```
HTREEITEM hti_res;
hti_res=FindItemData( hti, dwData);
if(hti_res!=NULL )
    return hti_res;
```

2) 应用 LogiScope 进行软件质量度量

为了更好地了解单元的代码质量,对单元形成全面的评价,我们可以使用 LogiScope 进一步对软件的静态质量进行度量。LogiScope 的质量度量模型主要依据 ISO/IEC9126(GB/T 16260)软件质量模型。对于软件的质量度量主要按如下步骤进行:

① 确定测试所采用的质量模型(C/C++ /Java)。

② 确定测试方法。

③ 执行测试。

④ 测试结果分析汇总。

在 HRMIS 的测试中，我们主要使用 LogiScope 的 C++ 质量度量模型进行度量，具体的执行步骤如下：

① 建立软件质量度量工程文件，选择代码所在的目录。

② 选择质量度量模型，可以使用系统默认的质量度量模型（工具安装目录\LogiScope\Ref\LogiScope. ref），也可以使用用户自定义的质量模型。用户可以通过修改 LogiScope. ref 文件创建自己的质量模型。

③ 通过以上步骤可以建立软件质量度量工程，工程建立后对程序进行编译处理，然后启动浏览器（start Viewer），可以看到具体的度量结果。通过 Viewer 可以看到对函数、类以及系统的静态质量分析。

HRMIS 的员工管理模块 CFVClerk 这个类的度量结果如下：

(1) 调用关系图。调用关系图（如图 11-12 所示）中主要标明函数或者类之间的调用关系，调用关系可以分为直接调用、间接调用和递归调用（在 LogiScope 中黑色表示直接调用、蓝色表示间接调用、红色表示递归调用）。

应用调用关系图可以对被测对象的结构做出分析，用以判断被测对象的函数/类之间的调用是否符合预先的设计，是否存在递归调用关系等（参见 10.2.3 节静态分析内容）。

递归调用是要避免的，因为递归调用会引起程序的无限循环从而导致系统资源耗尽出现死机等现象。如图 11-12 所示的是 HRMIS 中类 CFVClerk、CClerkChange、CDlgTrain、CDlgClerk、CDlgPact、CDlgWork 等的调用关系图，可以看出这些类之间不存在递归调用的关系，直接调用和间接调用也符合其系统设计说明。

(2) 质量度量在 LogiScope 中，质量度量的结果主要通过基维亚特（Kiviat）图（如图 11-13 所示）的形式表现，基维亚特图主要表现为一个归一化的圆，度量值落在两个圆环之间标明符合质量标准，度量值落在两个圆环之外标明不符合质量标准，对于 CFVClerk 类存在如下的不符合度量标准的项目：

① FAN_INclass。类的扇入值，反映了类通过其数据成员和成员函数参数而引进的数据输入量，其值越大该类日后需要频繁修改的可能性就越大。计算公式：FAN_INclass＝cl_data_prot＋cl_data_publ＋cl_usedp＋cl_data_vari

其中 cl_data_prot：类中 protected 类型数据成员的数量；cl_data_publ：类中 public 类型数据成员的数量；cl_usedp：类函数所使用的参数的数量总和；cl_data_vari：类函数中使用的本类的属性次数总和）。正常值在 0～15 之间，本类的度量值为 54 超出标准。

② Cl_dep_meth。类对其他模块的依赖性，主要计算必须依赖本类之外的其他模块才能实现其功能的类函数的个数。标准值在 0～6 之间，本类度量值为 19 超出标准。

③ Cl_wmc。类中各个成员函数的圈复杂度的总和，反映了函数的复杂程度。标准值在 0～25 之间，本类度量值为 61，超出度量范围。

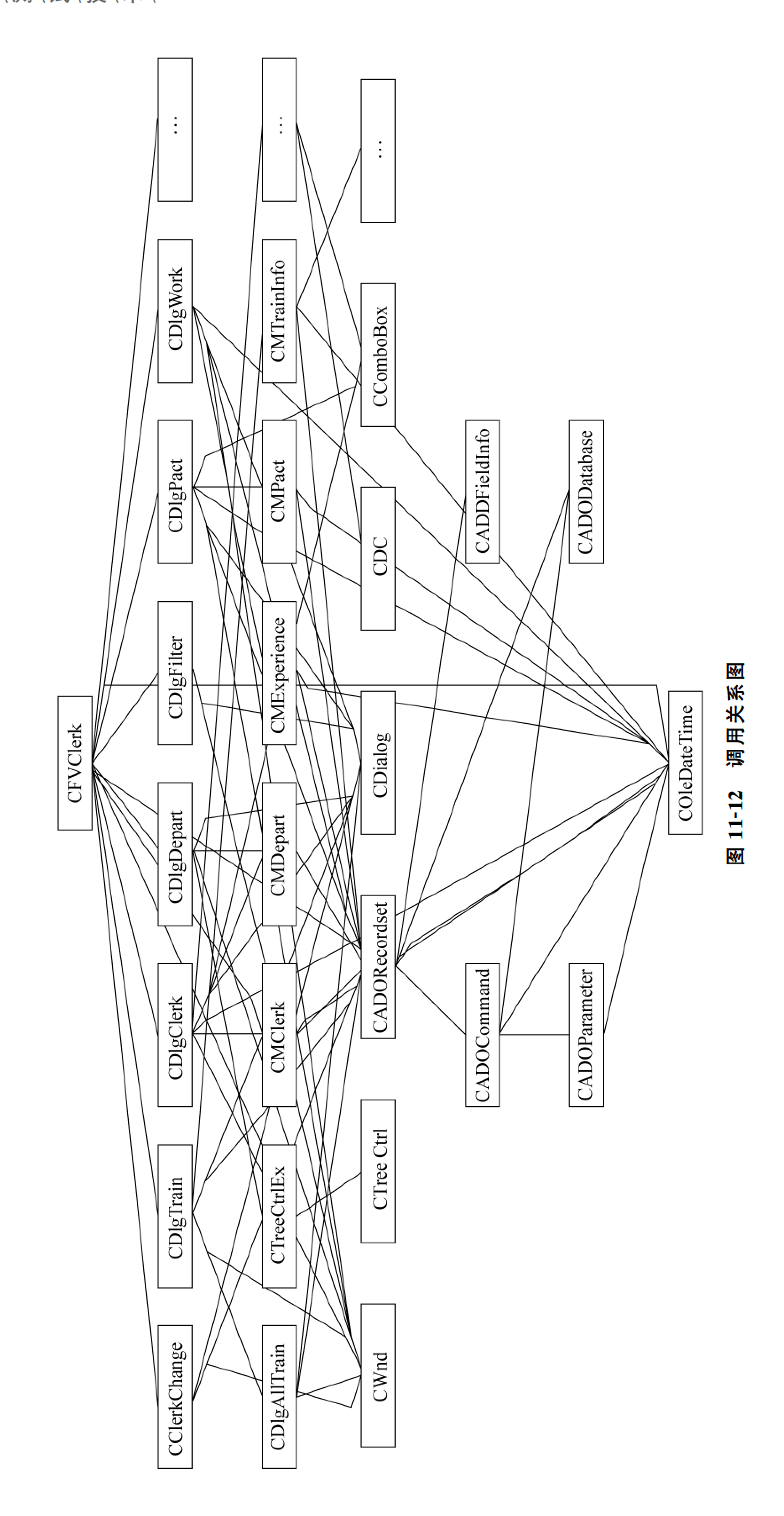
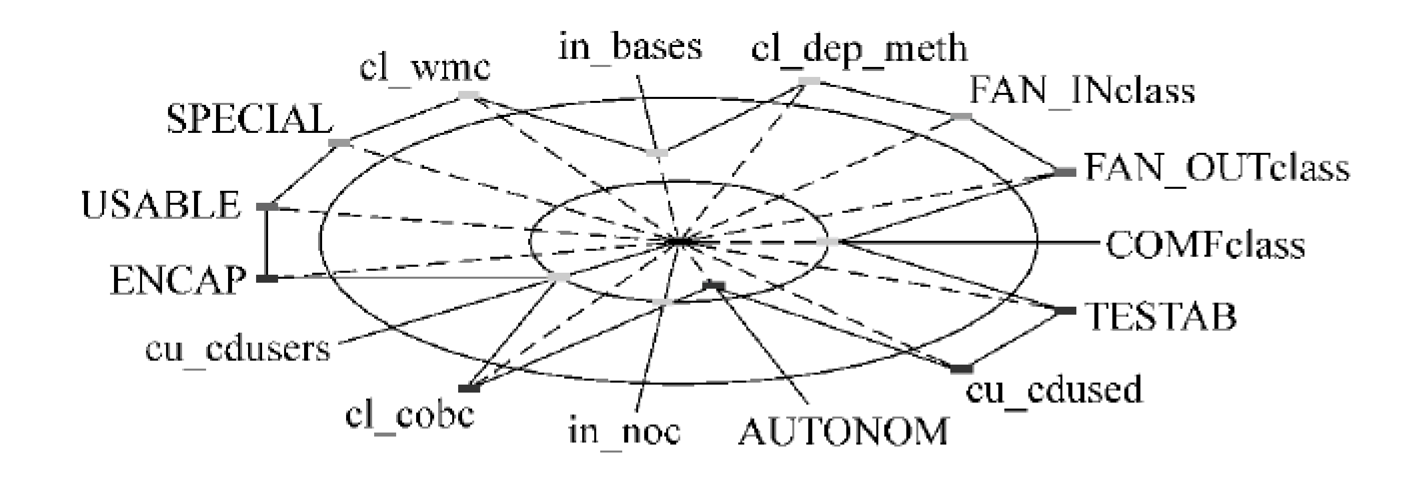

…
CDlgWork
CDlgPact
CDlgFilter
CFVClerk
CDlgDepart
CDlgClerk
CDlgTrain
CClerkChange
…
CMTrainInfo
CMPact
CMExperience
CMDepart
CMClerk
CTreeCtrlEx
CDlgAllTrain
…
CComboBox
CDC
CDialog
CADORecordset
CTree Ctrl
CWnd
CADDFieldInfo
CADOCommand
CADODatabase
CADOParameter
COleDateTime

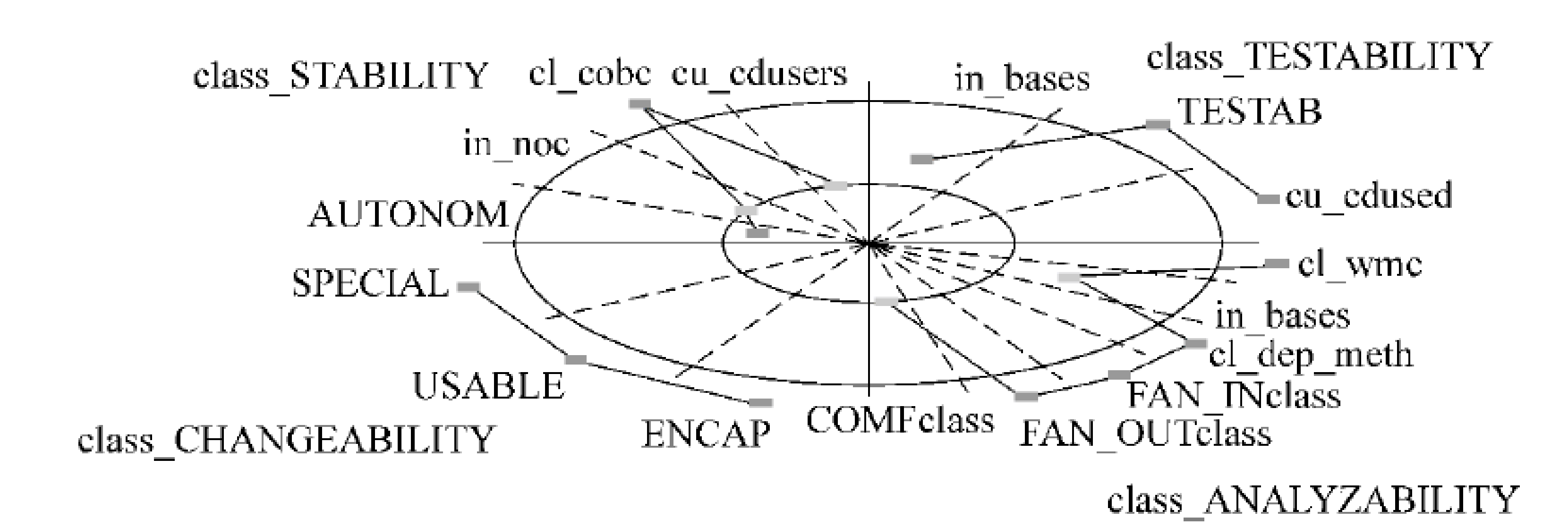
图 11-12　调用关系图



AXIS	LOW	HIGH	VALUE
COMFcalss	.20	+00	74
FAN_OUTclass	.00	20.00	29.00
FAN_INclass	.00	15.00	54.00
cl_dep_meth	.00	6.00	19.00
in bases	.00	3.00	1.00
cl_wmc	.00	25.00	61.00
SPECIAL	.00	25.00	45.00
USABLE	.00	10.00	18.00
ENCAP	.00	5.00	8.00
cu cdusers	.00	4.00	0.00
cl_cobc	.00	12.00	27.00
in_noc	.00	2.00	0.00
AUTONOM	30.00	100.00	18.18
cu cdused	.00	4.00	12.00
TESTAB	.00	100.00	170.00

图 11-13 质量度量的 Kiviat 图

在 HRMIS 的 CFVClerk 类的分析结果中存在多个不符合标准的度量项目(如图 11-13 所示，并未一一列举)，因此本类的综合度量结果较差，综合评定如下图 11-14 所示。



CRITERION	CLASS
class_TESTABILITY	FAIR
class_STABILITY	FAIR
class_CHANGEABILITY	POOR
class_ANALYZABILITY	FAIR
SYNTHESIS	POOR

图 11-14 CFVClerk 综合评定

(3) 主要函数控制流程图。Logiscope 可以对类中的方法(函数)进行单独分析,这对于一些算法复杂的函数来说是很有帮助的。在本例中,选择 CFVClerk 的消息函数 OnButtonClicked(如图 11-15 所示)进行控制流分析。

```
CFVClerk
  AssertValid()
  CFVClerk()
  ~CFVClerk()
  DoDataExchange(CDataExc
  Dump(CDumpContext &dc)
  FillListCtrl(CXListCtrl *pList,
  FindString(CStringArray *mS
  InitListCtrl(CXListCtrl *pList)
  OnBeginlabeleditListClerk(N
  OnBtnAdd()
  OnBtnCondition()
  OnBtnDel()
  OnBtnQcy()
  OnBtnViewchange()
  OnButtonClicked(WPARAM,
  OnCheckbox(WPARAM, LPA
  OnComboSelection(WPARAI
  OnEditEnd(WPARAM, LPAR/
  OnEndlabeleditListClerk(NM
  OnInitialUpdate()
  SetNewDepart(CString strCo
  m_dlgChange
  m_dlgClerk
  m_dlgDepart
  m_dlgFilter
  m_dlgPact
  m_dlgTrain
  m_dlgWork
  m_Male
```

```
/////////////////////////////////////////////////////////////////////////////
// OnButtonClicked
//
// This method shows how to handle WM_XLISTCTRL_BUTTON_CLICKED messages
// from XListCtrl
//
LRESULT CFVClerk::OnButtonClicked(WPARAM wParam, LPARAM lParam)
{
    int nItem    = (int) wParam;
    int nSubItem = (int) LOWORD(lParam);
    UINT nID     = HIWORD(lParam);

    if (nID != IDC_LIST_CLERK) {
        return -1;
    }

    if (nItem >= 0 && nSubItem >= 0)
    {
        CString strButtonText = m_xlistClerk.GetButtonText(nItem, nSubItem);

        // set text (if any) to red
        COLORREF crText, crBackground;
        m_xlistClerk.GetItemColors(nItem, nSubItem, crText, crBackground);
        CString strText = m_xlistClerk.GetItemText(nItem, nSubItem);
        m_xlistClerk.SetItemText(nItem, nSubItem, strText,
            RGB(255,0,0), crBackground);

        switch(nSubItem) {
        case 1://合同
            m_dlgPact.m_edCode       = m_xlistClerk.GetItemText(nItem,0);
            m_dlgPact.m_edDepartCode = m_xlistClerk.GetItemText(nItem,2);
            m_dlgPact.DoModal();
            break;
```

图 11-15 CFVClerk OnButtonClicked()

图 11-16 为 OnButtonClicked 成员函数的控制流程图,从图中可以看到本函数包含一个语句块,两个判断分支结构,一个带 Break 语句的 Switch 分支结构,可以通过与函数最初的设计说明进行比对,确定函数是否符合最初的设计要求。

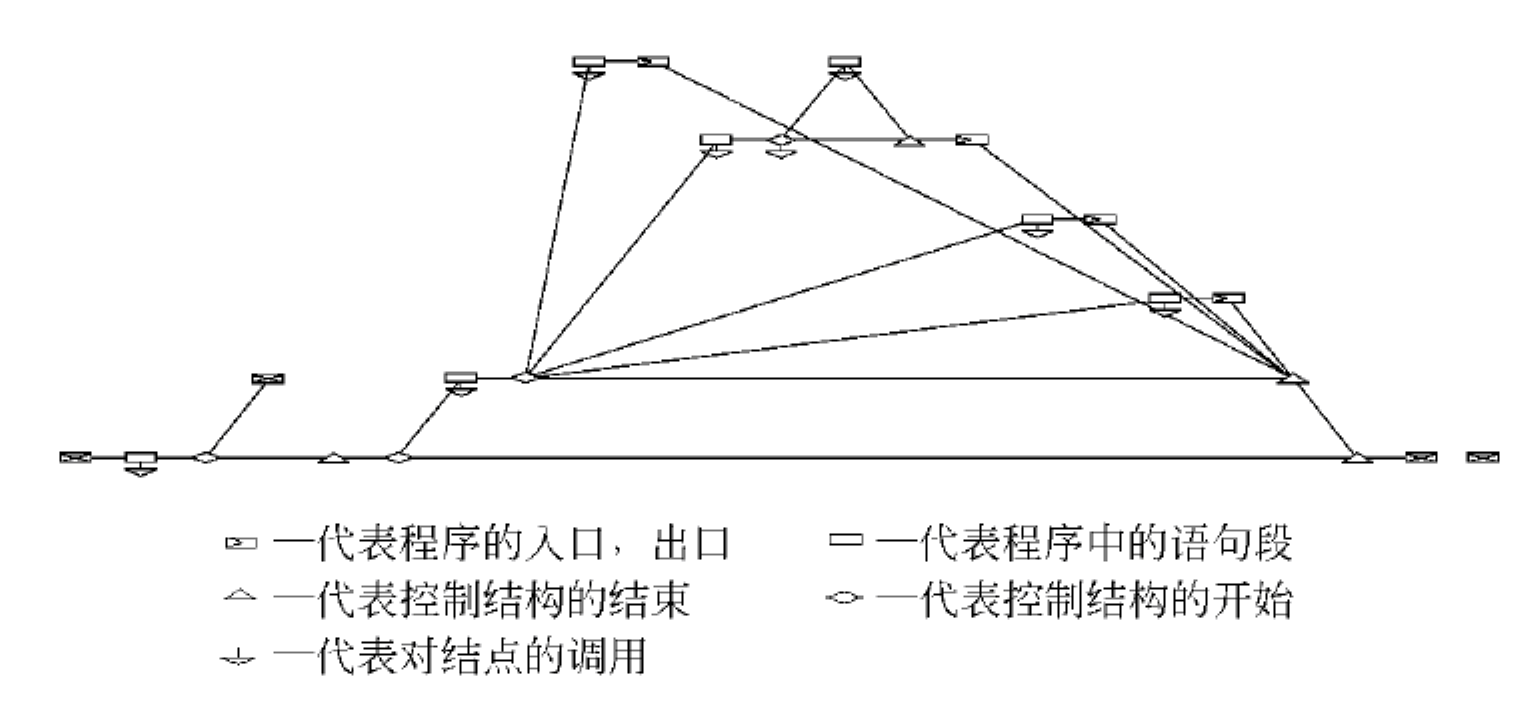


图 11-16 OnButtonClicked 控制流程图

11.5.3 集成测试执行情况

在单元测试完成后,我们应该对符合集成条件的单元尽快进行集成测试,以验证各个单元集成在一起后是否能够按照预先的设计运转。

10.3.2 节介绍了集成测试,并对 HRMIS 的集成测试做了规划设计,并编写了一个集成测试用例(用例 11-4)。在这里我们按照既定的自顶向下的集成测试方案,对类 CFVClerk、CMClerk 进行集成测试。

【用例 11-4】

<table>
<tr><td colspan="2">用例名称</td><td colspan="2">测试修改员工信息</td><td>用例标识</td><td colspan="2">P200901UT-YGGL-YGLR-04</td></tr>
<tr><td colspan="2">测试追踪</td><td colspan="5">—</td></tr>
<tr><td>用例说明</td><td colspan="6">测试修改指定员工信息,CMClerk 能否准确接收到新的信息:可供修改的信息有员工名称、姓名、性别、政治面貌等信息。</td></tr>
<tr><td rowspan="4">用例的初始化</td><td>硬件配置</td><td colspan="5">无特殊要求</td></tr>
<tr><td>软件配置</td><td colspan="5">软件可以正常启动运行,正常连接 MySQL 数据库</td></tr>
<tr><td>测试配置</td><td colspan="5">无特殊要求</td></tr>
<tr><td>参数设置</td><td colspan="5">部门组织结构已初始化</td></tr>
<tr><td colspan="7">操作过程</td></tr>
<tr><td>序号</td><td colspan="2">输入及操作说明</td><td>期望的测试结果</td><td colspan="2">评价标准</td><td>备　注</td></tr>
<tr><td>1</td><td colspan="2">在员工信息列表中选择一个员工并修改其各项信息(包括姓名、性别、年龄、出生日期、身份证号等)</td><td>—</td><td colspan="2">—</td><td></td></tr>
<tr><td>2</td><td colspan="2">确认修改</td><td>—</td><td colspan="2">—</td><td></td></tr>
<tr><td>3</td><td colspan="2">查看信息修改是否正确</td><td>—</td><td colspan="2">信息已成功修改并更新至数据库</td><td></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td></td><td colspan="2"></td><td></td><td colspan="2"></td><td></td></tr>
<tr><td colspan="3">前提和约束</td><td colspan="4">• 查询系统中已存在的员工且员工记录不少于 1 条
• 所输入的新的身份证号不能重复</td></tr>
<tr><td colspan="3">过程终止条件</td><td colspan="4">无</td></tr>
<tr><td colspan="3">结果评价标准</td><td colspan="4">信息已成功修改并更新至数据库</td></tr>
<tr><td colspan="3">设计人员</td><td>—</td><td colspan="2">设计日期</td><td>—</td></tr>
</table>

再来看一下这个用例,这个用例是用来验证 CFVClerk 能否将变更的员工信息准确传递给 CMClerk 类,为了验证方便,我们已经为接口函数增加消息输出。下面为这个用例构造一组测试数据:

姓名:谢静
性别:女
出生日期:1987-8-10
政治面貌:民主同盟
文化水平:博士
婚姻状况:
家庭住址:桑园街 6 号
身份证号:370902198708109239

办公电话：86515187
移动电话：—
紧急联系人：—
联系方式：—
毕业院校：—
入职时间：2008-12-14
岗位职责：—

本例测试家庭住址的变更，家庭住址由现在的“桑园街 6 号”变更为“高新开发区雅居园 1—302”。

我们将 CFVClerk 和 CMClerk 集成编译后，运行 HRMIS，输入以上测试数据保存员工信息，然后对该员工记录的家庭住址信息进行修改。操作完成后，我们得到如图 11-17 的测试结果。可以看到，我们设置的断言给出的信息和我们提供的数据信息是一致的。为了进一步验证，查看一下 HRMIS 的后台数据库信息（如图 11-18 所示），据此可以判断 CFVClerk 和 CMClerk 集成后可以完成员工信息的修改。



图 11-17　信息输出

```
mysql> select YGXM,JTZHZH from T_YG where YGXM like '谢%';
+------+----------------------+
| YGXM | JTZHZH               |
+------+----------------------+
| 谢静 | 高新开发区雅居园1－302 |
+------+----------------------+
1 row in set (0.00 sec)

mysql> _
```

图 11-18　数据查询结果

11.5.4　系统测试执行情况

系统测试是软件交付客户前由开发方所进行的最后一道测试程序，系统测试的效果

直接影响软件系统交付的质量。前面已经介绍过，系统测试不仅关注软件的功能，同时也关注软件植入客户环境后作为一个系统的性能，如效率、可靠性、可移植性等。关于系统测试的执行，本书主要从功能测试和性能测试这两个应用最为广泛的角度讲解。

1. 功能测试执行

我们在第三章 3.3 节讲系统测试时，对于 HRMIS 的部分功能设计了部分用例，下面我们仍然以此为例看一下功能测试的执行过程。

1）手工执行

作为示例，选择用例 11-5～11-7 这三个用例来介绍一下测试用例的执行过程。让我们先来回顾一下这几个用例：

【用例 11-5】

用例名称	增加员工记录		用例标识	P200901UT-YGGL-YGLR-05
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	增加一条记录，“姓名”输入“张三”，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据构成划分的有效等价类			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“张三”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息成功保存	可以查询到新录入的员工信息	
前提和约束	所输入的员工身份证号不能与已有员工信息重复			
过程终止条件	无			
结果评价标准	查询到的员工信息与录入信息保持一致			
设计人员	—	设计日期	—	

【用例 11-6】

用例名称	增加员工记录	用例标识	P200901UT-YGGL-YGLR-06	
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	增加一条记录，“姓名”输入 abc，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据构成划分的有效等价类			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		
操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“abc”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息成功保存	可以查询到新录入的员工信息	
前提和约束	所输入的员工身份证号不能与已有员工信息重复			
过程终止条件	无			
结果评价标准	查询到的员工信息与录入信息保持一致			
设计人员	—	设计日期	—	

【用例 11-7】

用例名称	增加员工记录	用例标识	P200901UT-YGGL-YGLR-7	
测试追踪	业务需求说明书：人事部门招聘专员对于新招聘的职员信息可以录入到 HRMIS 系统中，TR01			
用例说明	增加一条记录，“姓名”输入“!? <>”，其他输入项分别输入符合要求的数据后，单击“保存”按钮。用例设计方法：以数据构成划分的无效等价类			
用例的初始化	硬件配置	无特殊要求		
	软件配置	软件可以正常启动运行，正常连接 MySQL 数据库		
	测试配置	无特殊要求		
	参数设置	部门组织结构已初始化		

续表

操作过程				
序号	输入及操作说明	期望的测试结果	评价标准	备　注
1	打开员工管理界面	显示员工信息列表	员工信息列表中显示已存在的员工信息	
2	单击“新增”按钮	打开员工信息输入窗口	显示员工信息输入窗口	
3	在姓名栏输入“!? <>”，其他数据栏输入相应信息	—	—	
4	单击“保存”按钮	员工信息保存失败	系统给出错误提示，查询不到新录入的员工信息	
前提和约束		所输入的员工身份证号不能与已有员工信息重复		
过程终止条件		无		
结果评价标准		查询不到新录入的员工信息且对原有数据信息无影响		
设计人员		—	设计日期	—

通常，在进行系统测试之前，我们首先应该对所有的测试用例进行分析，统筹安排，使相互关系紧密的用例聚集在一起，这样可以在同一阶段对系统的某个部分进行集中测试，易于发现问题和节约资源。我们把这样一个测试集称为“测试套件”。比如，上面这三个用例都是测试 HRMIS 的“增加员工记录”这个功能的，也就是说这三个用例是功能相关的。那么我们就可以把三个用例作为一个测试套件，交给一个测试人员来集中执行。划分测试套件可以根据测试需要进行，一般来讲可以以功能相关、环境相关等因素进行考虑。

（1）功能相关。功能相关是最常见的一种方式，这个很好理解，除了我们上面说到的可以把测试同一个功能的用例作为一个测试套件之外，另外还可以考虑与一个功能具有紧密关系的上下游功能的测试用例，这样做可以免去下游用例准备初始数据的步骤。比如上面这组用例是测试“增加员工记录”的，那么我们可以联想到“修改员工信息”这组测试用例，这两组用例放在一起，恰好形成一组前后相关，可以中继使用的数据信息链，它们聚集在一起形成一个更大的测试套件。测试套的划分，还要考虑测试人员的工作量适合，不能无限度聚集。

（2）环境相关。环境相关是指某些测试项目中对环境有特殊依赖，这样我们可以把这样的一些用例抽离出来，形成一组要求同一环境测试的测试用例集，也就是测试套件。比如，在 HRMIS 中，假设我们要测试考勤、考评这两项功能，则需要一个外部设备——考勤机。考勤机不属于 HRMIS 研发单位的，且 HRMIS 很可能被要求支持多个厂家的设备，那么这些考勤机可能就需要从各个厂家临时借来用以系统联调。这就要求，测试组把测试和考勤机相关的测试用例单独抽离出来，集中在一段时间进行，此种情形就构成一个

环境相关的测试套件。

手动执行测试的过程就是动态运行 HRMIS,按照测试用例的设计步骤和数据操作软件,观察软件的响应,并记录和报告所观察到的现象(或称事件)。关于测试记录、事件报告、日志和问题报告单的格式我们在前面已经讨论过,在此不再赘述。下面,我们着重看一下依托测试管理系统的测试执行过程。

目前,很多测试管理系统都支持用例的设计和执行,在本案中我们使用 TestDirector(为了叙述方便以下简称 TD)作为测试管理平台,通过 TD 统一管理测试用例的执行。

(1) TD 中的测试用例。上面所编制的测试用例可以在 TD 中实现,如用例 11-5 在 TD 中对应的用例是“新增员工记录 P2009ST-YGGL-YGLR-01”,用例 11-6 对应的是“新增员工记录 P2009ST-YGGL-YGLR-02“,用例 11-7 对应的是“新增员工记录 P2009ST-YGGL-YGLR-03“(如图 11-19 所示)。



图 11-19　TD 中的测试用例

其中,左边栏表示测试套件(本例中是以功能区分的测试套件)和用例标识,右部区域是测试用例的详细内容区,图中显示的是测试用例的步骤。

测试用例可以追踪到测试需求,如图 11-20 所示,单击 Select Req,会在右边显示我们在 TD“REQUIREMENT”中录入的测试需求,通过鼠标就可以方便地设定用例所覆盖的测试需求。

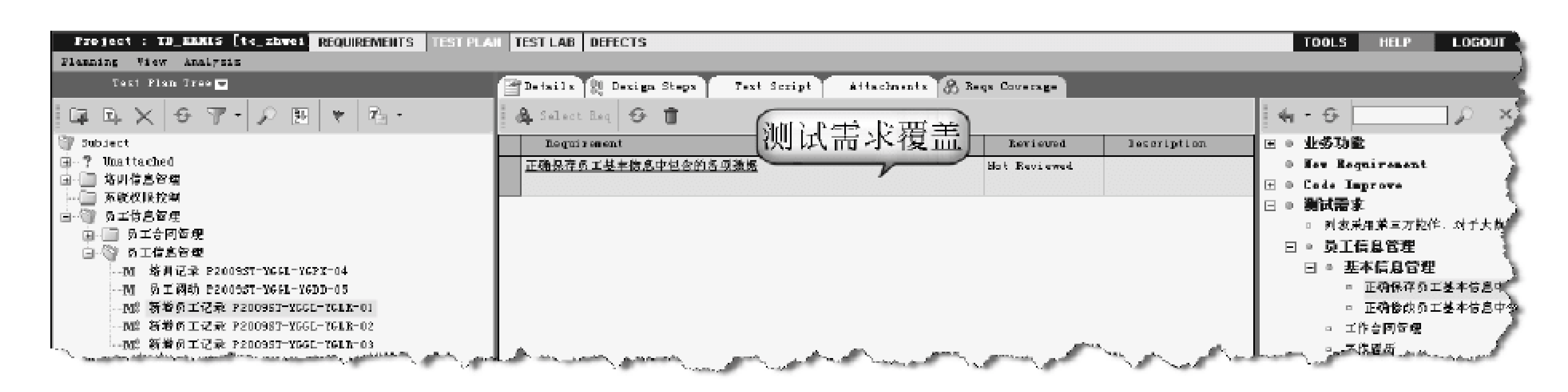


图 11-20　测试需求覆盖选择示意

(2) 测试机调度。测试用例的执行是在 TD 的 TESTLAB(测试实验室)栏进行。

通过 TD,测试负责人可以统一调度所有的测试机,下面我们来看一下测试机的管理和调度过程。

单击 TESTLAB 栏的菜单 HOST 可以打开主机管理界面(Host Manager)(如图 11-21 所示),在这里,我们可以将所有的测试机加入到主机列表,或者单击“Get Net”,TD 会自动搜索局域网内的机器加入到主机列表。单击右边的 Create,我们可以创建 HRMIS 的

测试项目的测试设备(如图 11-22 所示)。HRMIS 主机组(Host Group)创建后,即可从主机列表中选择指定的设备。

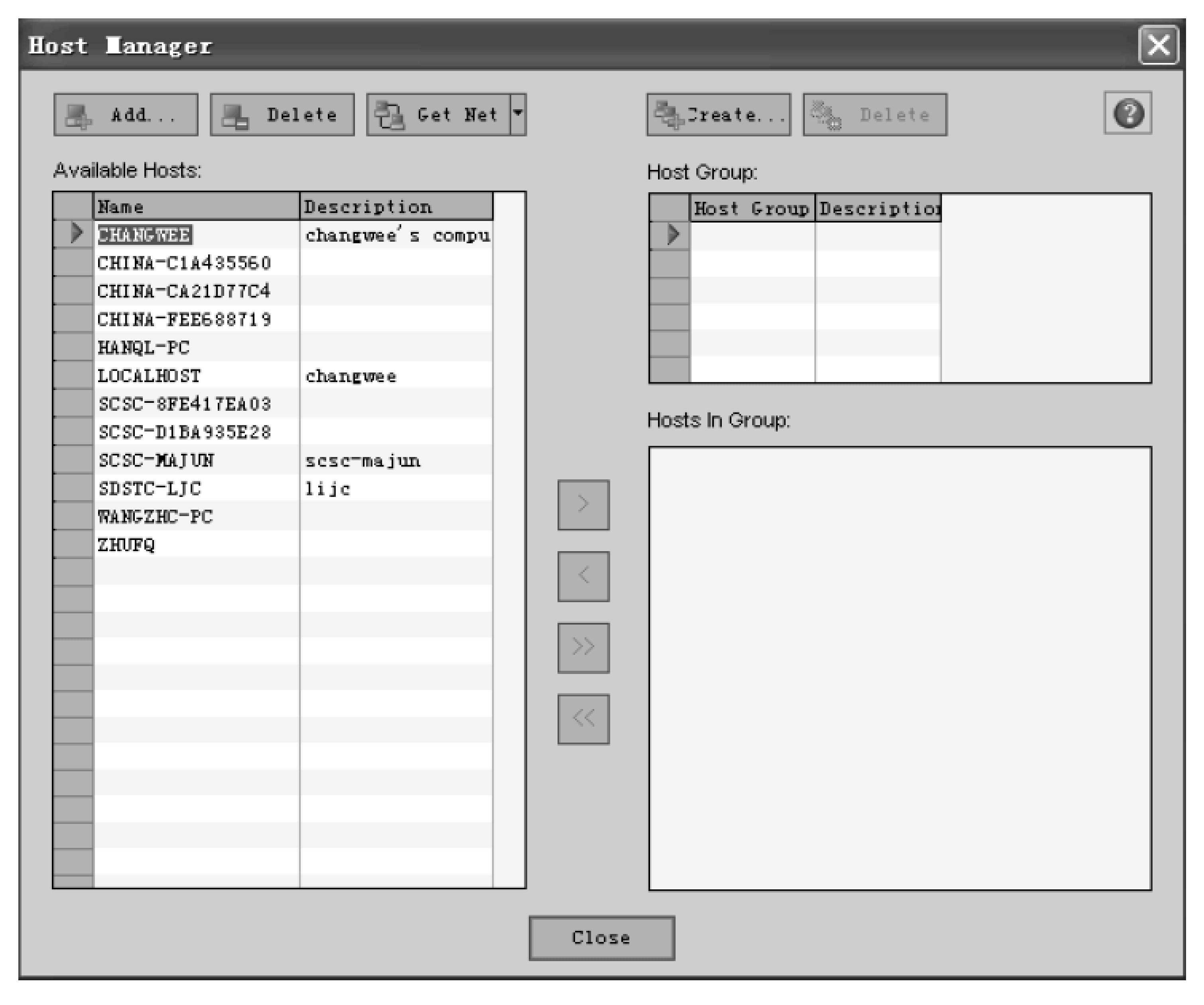


图 11-21　测试设备管理

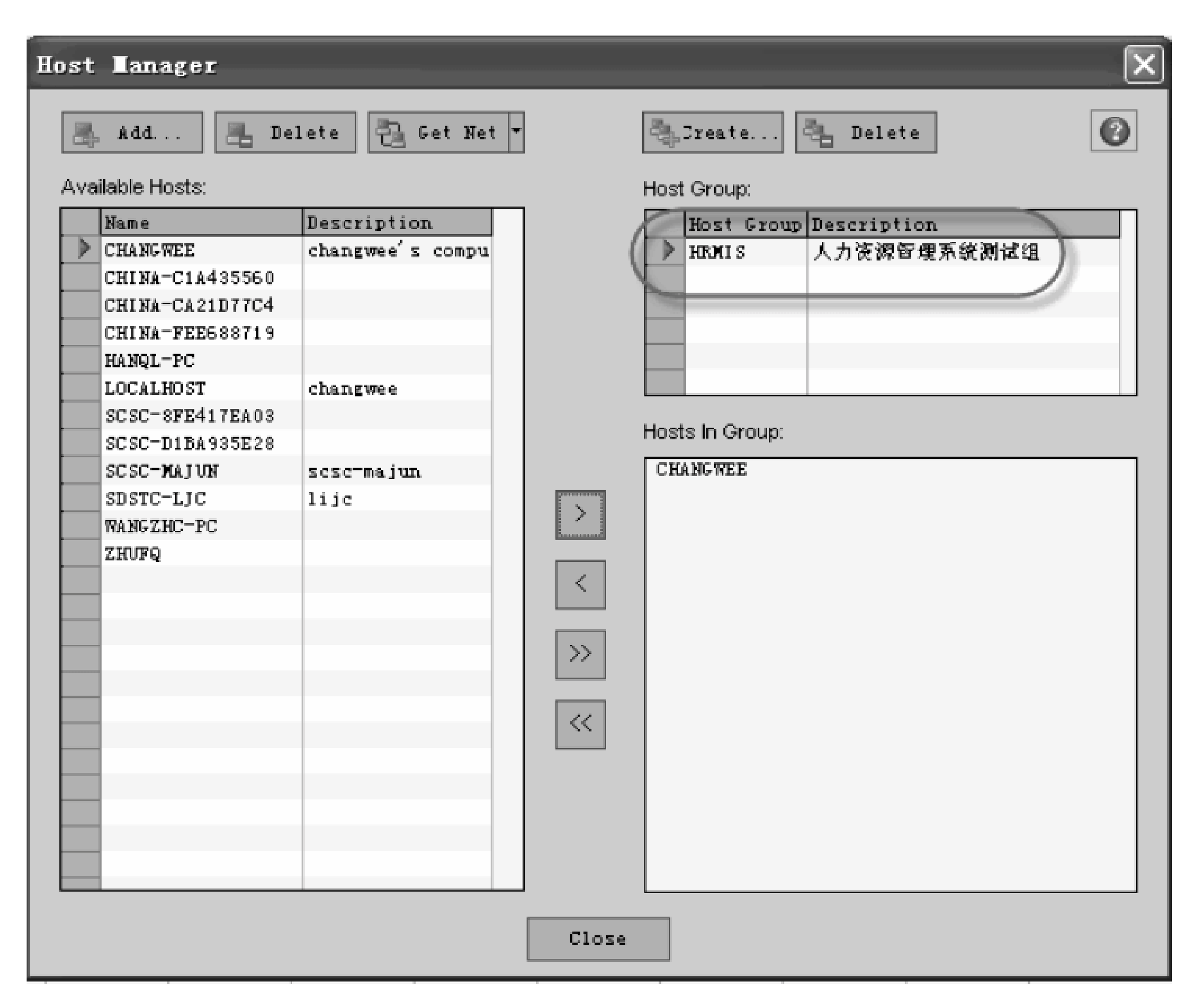


图 11-22　创建 HRMIS 的测试设备组

(3)测试用例分配。将测试设备纳入到测试项目后,接下来,测试负责人可以分配测

试任务，设定测试执行者(Responser Tester)，并将测试套件分配到指定机器上。

如图 11-23 所示，单击 Run Test Set，TD 会弹出手工测试选择框(Manual Test Run)，我们选择第二项 Execution dialog box 即可进入测试套件的用例分配界面(如图 11-24 所示)。

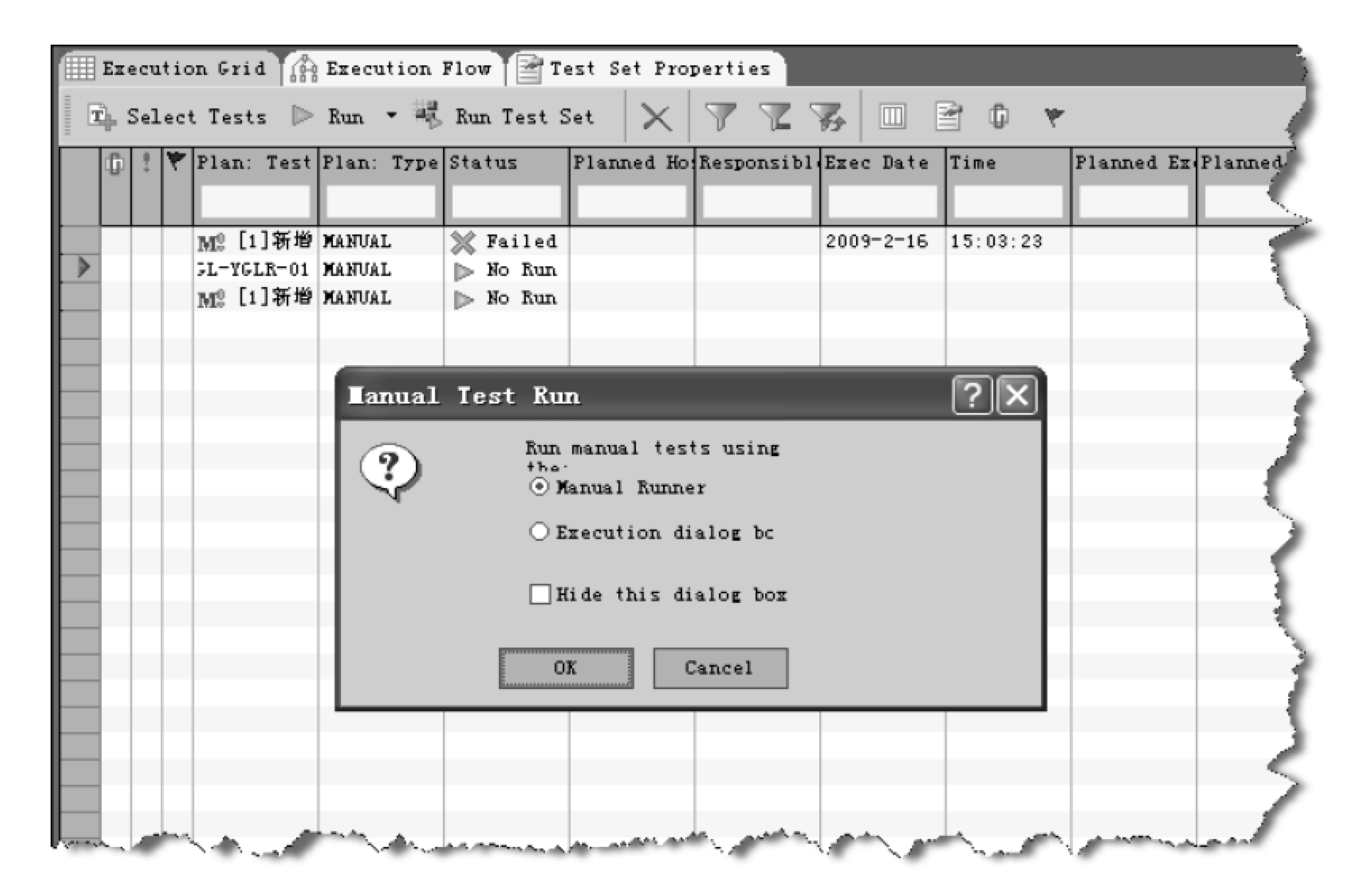


图 11-23　选择执行方式

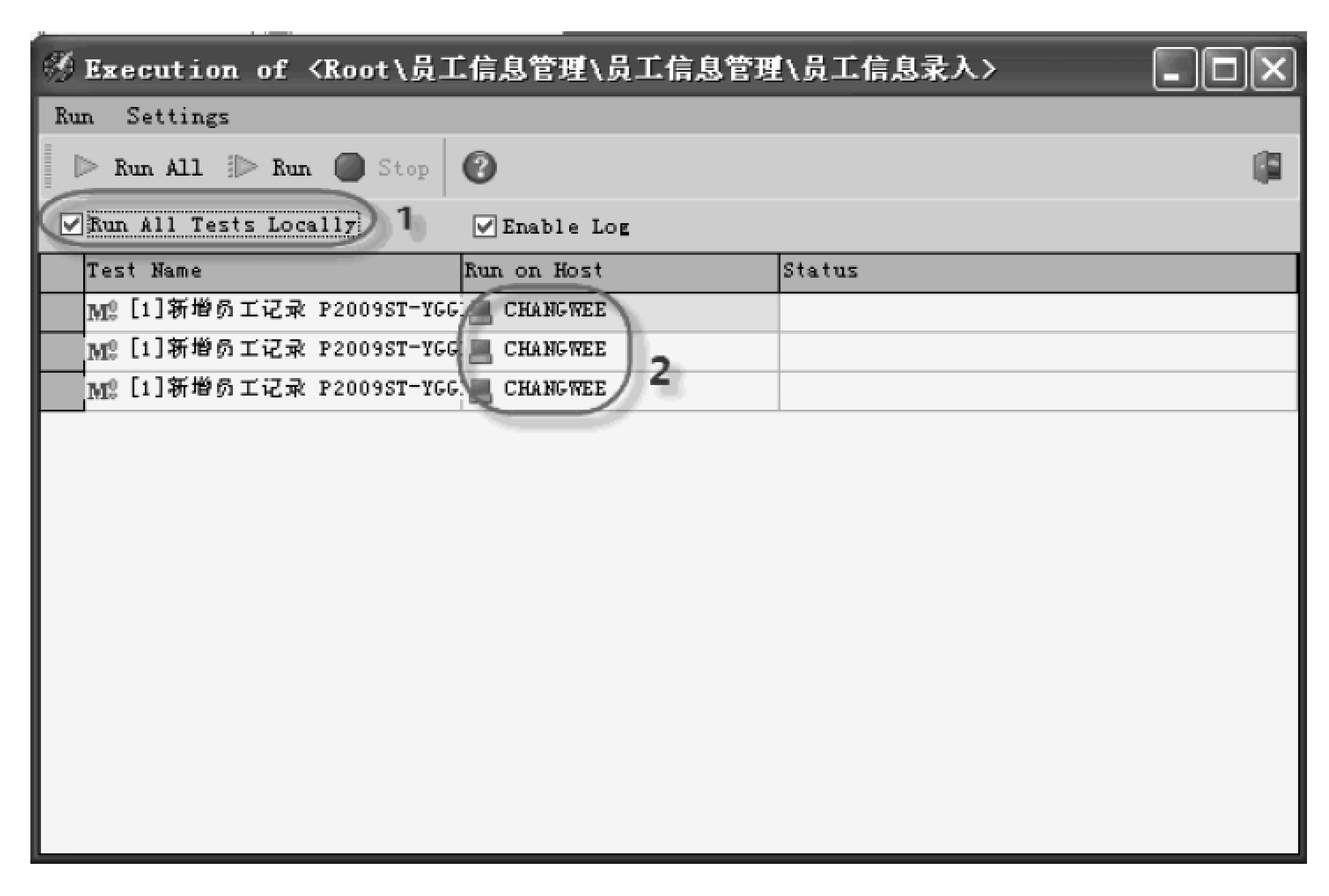
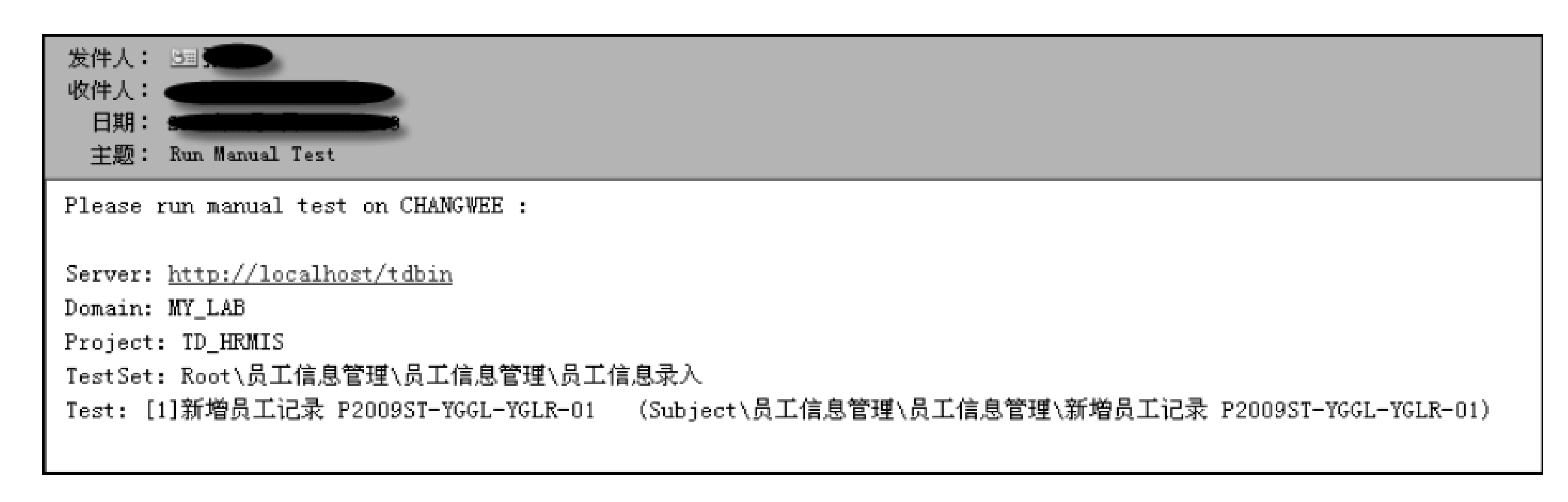


图 11-24　测试用例执行分配

这里，将"新增员工记录 P2009ST-YGGL-YGLR-01"、"新增员工记录 P2009ST-YGGL-YGLR-02"、"新增员工记录 P2009ST-YGGL-YGLR-03"三个测试用例都分配到 changwee 这个测试机上(选择"Run All Tests Locally"，TD 会自动将当前所选的用例分配给本机)，测试人是 tc_zhwei。

测试负责人分配测试任务后，TD 会自动向测试执行人员发送邮件通知（如图 11-25 所示）。


发件人：[redacted]
收件人：[redacted]
日期：[redacted]
主题：Run Manual Test

Please run manual test on CHANGWEE :

Server: http://localhost/tdbin
Domain: MY_LAB
Project: TD_HRMIS
TestSet: Root\员工信息管理\员工信息管理\员工信息录入
Test: [1]新增员工记录 P2009ST-YGGL-YGLR-01 (Subject\员工信息管理\员工信息管理\新增员工记录 P2009ST-YGGL-YGLR-01)

图 11-25 测试执行通知邮件

（4）测试用例执行过程

我们看一下 TD 中的测试用例执行过程。本案中，测试人员 tc_zhwei 接到测试通知后登录 TD，选择测试用例“新增员工记录 P2009ST-YGGL-YGLR-01”，单击“执行”（run），进入用例执行界面（如图 11-26 所示）。

Manual Runner Test Set: <员工信息录入> Test: <[1]新增员工记录 P2009ST-YGGL-YGLR-01>

Exec Steps | End of Run | Cancel

Run Details

Run Name: Run_3-5_15-44-42 | Operating System Info
Tester: tc_zhwei
Status: Not Completed
Exec Date: 2009-3-5
Exec Time: 15:44:42

Test Details

Name: 新增员工记录 P2009ST-YGGL-YGLR-01 | More...

增加一条记录，姓名输入张三，其他输入项分别输入符合要求的数据后，单击保存。
用例设计方法：以数据构成划分的有效等价类

图 11-26 测试用例手工执行

用例的执行状态在 TD 中设置了 5 种，分别是 No Run、N/A、Failed、Pass 和 Not Complete，分别是未执行、不适用、失败、通过和未完成。测试步骤的设置应根据实际测试过程中所观察到的现象手工设置，如果是不适用，则可能导致测试用例的重新修订或者舍弃，如果是失败，则可能导致一个缺陷的发现。

在测试执行的过程中，每一步在设置为“Passed”之后，会自动跳转到下一个执行步骤，直至所有的测试步骤操作完成，如果全部步骤执行完后，与预期的结果一致且符合所设定的评价标准，则我们说这个测试用例通过了（见图 11-27）。图 11-28 是用例“新增员工记录 P2009ST-YGGL-YGLR-01”的执行结果。

按照同样的方法，我们执行用例“新增员工记录 P2009ST-YGGL-YGLR-03”，得到这

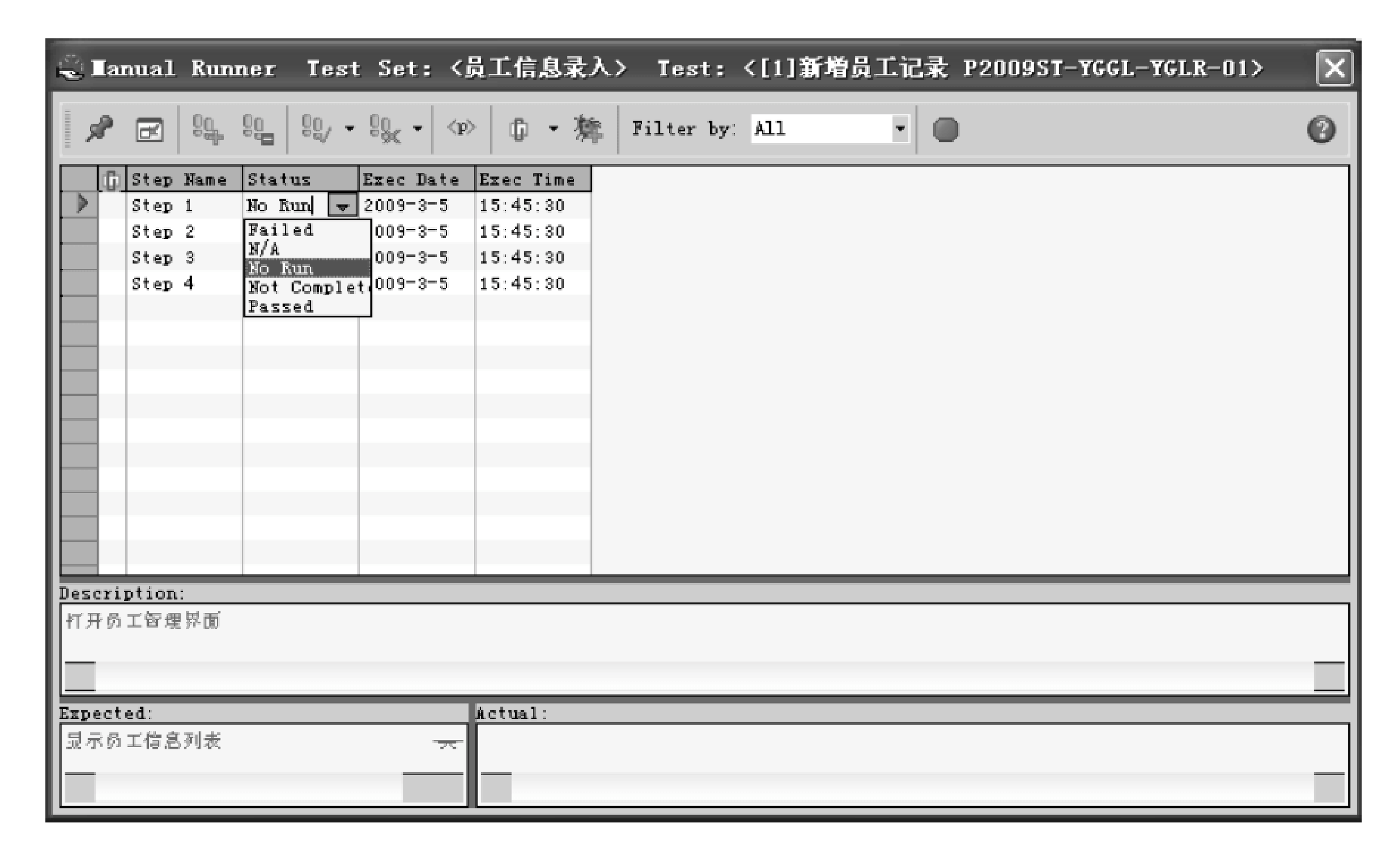


图 11-27　测试用例执行结果

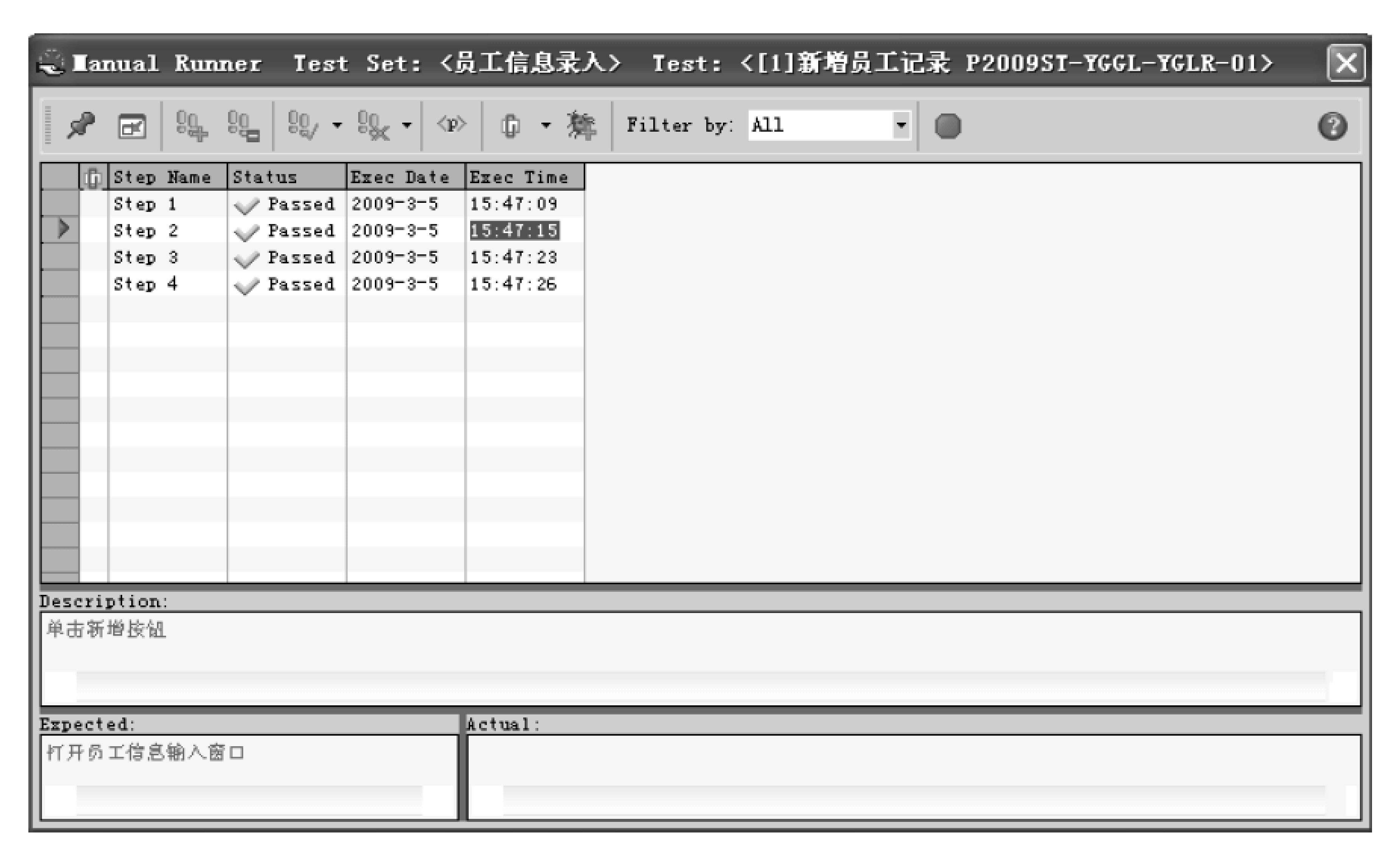


图 11-28　HRMIS 01 号用例执行结果

样一个结果(如图 11-29 所示),可以看到在这个用例的最后一个步骤出现了一个 Failed,这表明最后一步的测试结果与预期结果不一致,不符合设定的评价标准。我们将这个缺陷记录到 TD 中(严格地说,未经确认前,还不能称之为是一个缺陷)。

在缺陷报告时,TD 会自动将测试用例的执行步骤带入缺陷的描述信息中,我们将该步骤的实际观察到的结果补充到 Actual 栏即可。同时,要补充缺陷概述(summary)、缺陷发现人(Detected By)、缺陷发现日期(Detected on Date)、缺陷严重程度(Sererity)、缺陷所在模块(Project)等一些必要的信息(如图 11-30 所示)。



图 11-29　HRMIS 03 号用例执行结果

Add Defect

Clear　Attach:

Defect Information:

* Summary　新增员工记录，姓名为标点时可以保存成功

Page 1

* Detected By:	tc_zhwei	* Detected on Date:	2009-3-5
* Severity:	2-Medium	Assigned To:	
Detected in Version:	V0.2	Modified:	
Priority:		Project:	培训信息管理
Reproducible:	Y	Status:	New
Subject:	员工信息管理		

Description

Test Set:	员工信息录入
Test:	[1]新增员工记录 P2009ST-YGGL-YGLR-03
Run:	Run_3-5_15-57-3
Step:	Step 4

Description:
单击保存

Expected:
员工信息保存失败

Actual:
员工信息保存成功，且内容与所输入信息一致

Submit　Close

图 11-30　缺陷报告

这样一个完整的基于 TD 的测试用例执行过程就结束了，那么对于我们所发现的缺陷 TD 是如何处理的呢？缺陷管理是 TD 的一个核心模块，它对于缺陷的跟踪和处理基本上符合我们前面所述的有关缺陷管理的内容。例如，刚刚录入的这个缺陷（25 号缺陷），查看其在 TD 中的流转过程（如图 11-31 所示），我们可以看到：这个缺陷由测试人员

tc_zhwei 录入，状态为 new，admin 确认之后 25 号缺陷的状态被置为 open，并分配给开发人员 hanmj 进行处理。Hanmj 修改后，将 25 号缺陷的状态置为 fixed。tc_zhwei 对这个缺陷进行验证，验证通过后将其关闭(closed)(如图 11-32 所示)。

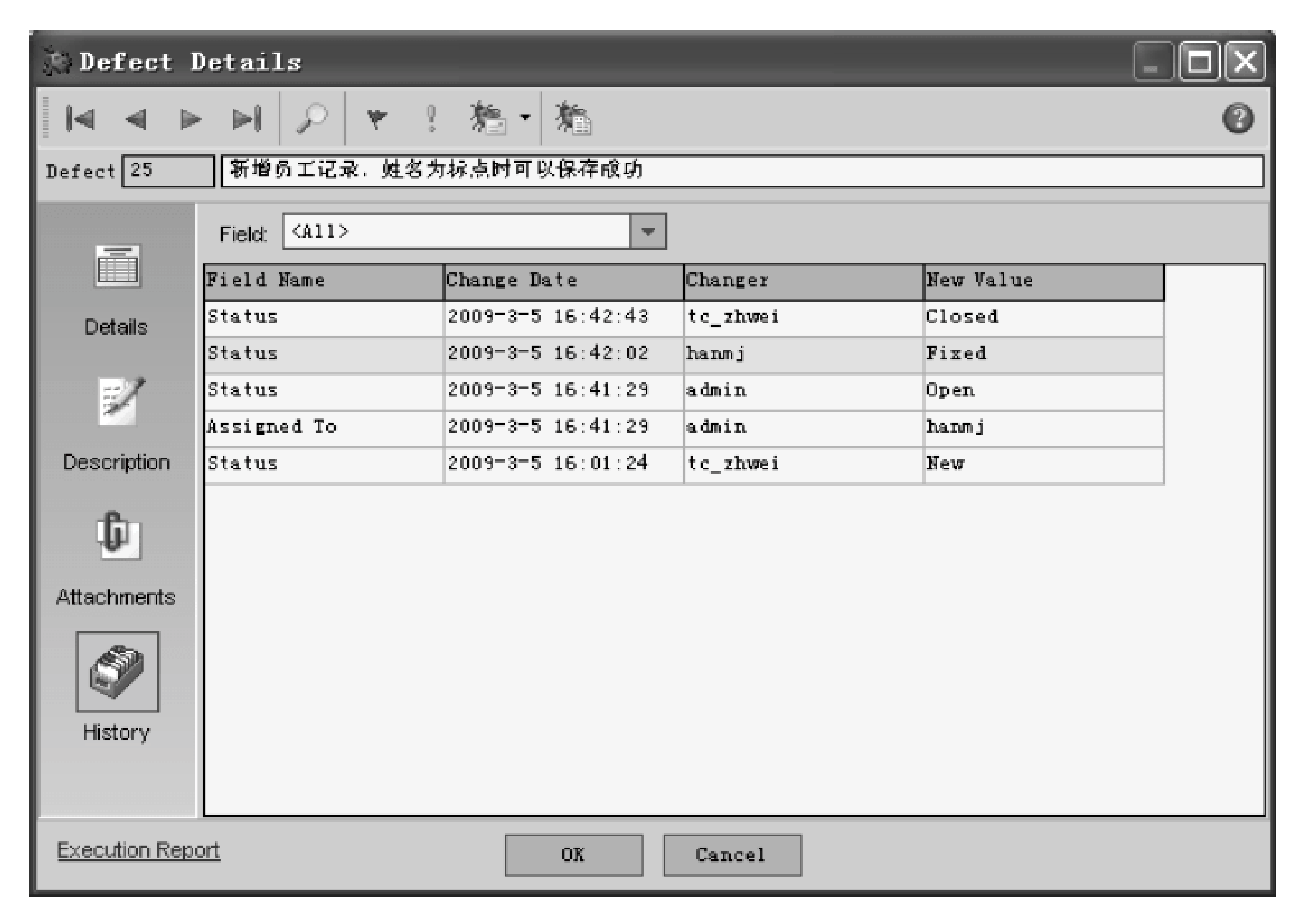


图 11-31　25 号缺陷状态变更历史记录

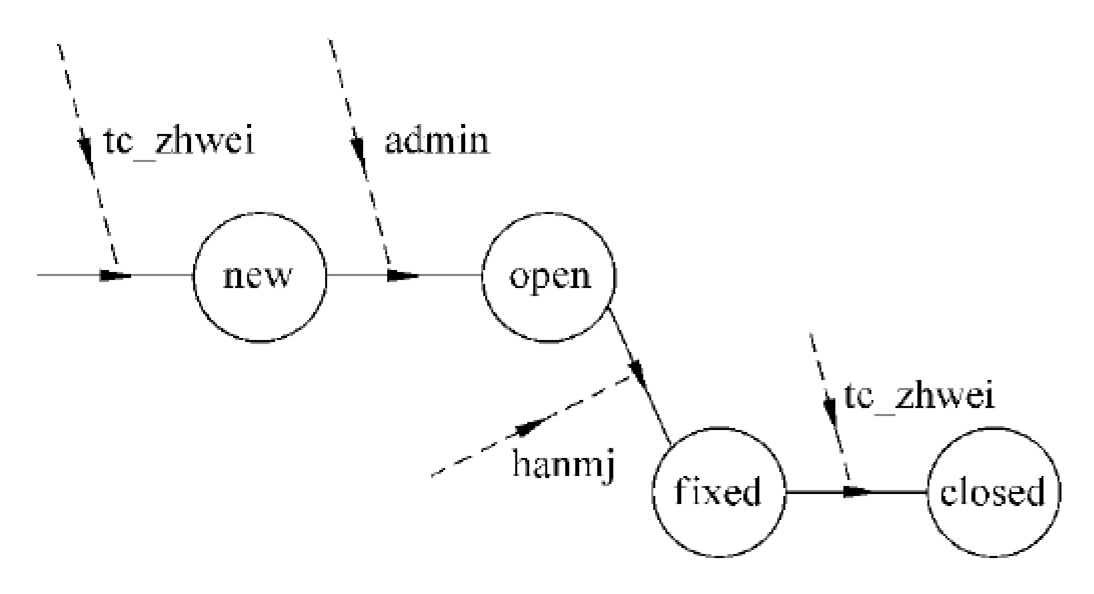


图 11-32　25 号缺陷处理流程

2) 自动执行

软件测试是发现问题、解决问题、验证问题这样一个反复的过程，因此关于自动化测试的研究和讨论从未停止过。通过测试自动化执行一方面可以提高缺陷的验证效率，同时也可以降低测试成本。

针对 HRMIS 的测试，可以开发测试脚本，使“员工信息管理”中的“增加员工”的测试自动化执行。这个自动化测试脚本的开发目前有多种测试工具支持，作为范例，我们在本案中分别应用 WinRunner 和 QuickTest Professional 来开发这个测试脚本。

(1) 应用自动化测试工具 WinRunner

① 录制测试脚本。首先运行人力资源管理系统(HRMIS)，然后启动 WinRunner，选择“Visual Basic”插件，并设置 GUI Map File 的模式为 GUI Map File per Test，脚本录制

模式选择 Context Sensitive。

WinRunner 有录制和手工编写两种脚本开发方式，并且有丰富的内置测试脚本语言(TSL)，能够实现丰富灵活的测试控制。在这里，我们选择录制模式生成测试脚本。录制测试脚本只需单击录制按钮，然后切换到 HRMIS 进行增加员工的有关业务操作即可(如图 11-33 所示)。

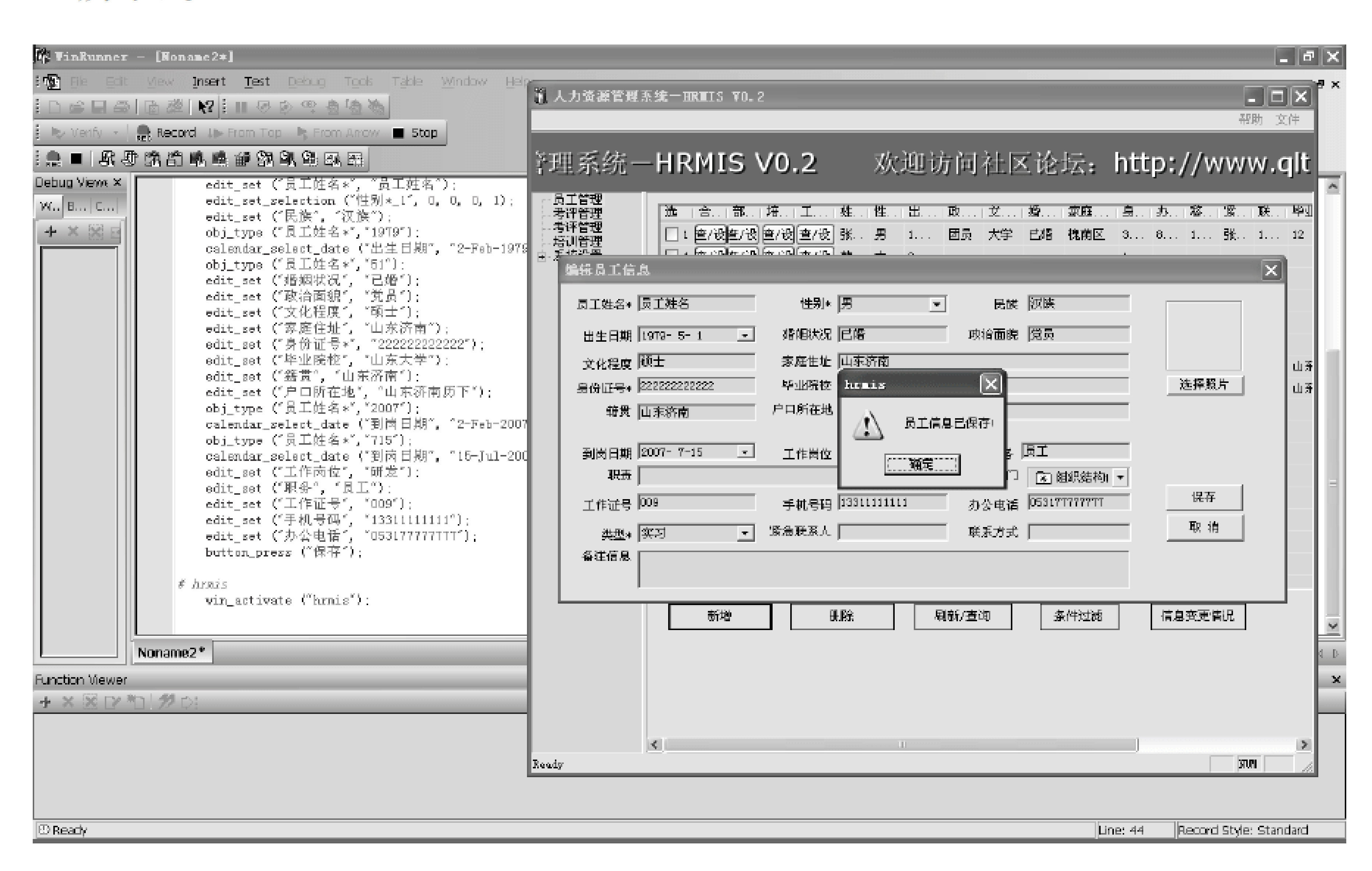


图 11-33　录制"增加员工"业务脚本

在使用 WinRunner 测试应用程序时，通常是通过检查 GUI 对象的属性，来判断所测功能是否正常，这种方式称之为 GUI 检查点。另外，WinRunner 还提供读取图像或非标准 GUI 对象上的文字的功能，通过手动撰写少量测试脚本，以检查文字是否正确，这种方式称之为文字检查点。本例的测试过程中，可以根据 HRMIS 本身的特点，应用 GUI 检查点来判断功能是否正常。在 HRMIS 中员工信息保存后，会弹出"员工信息已保存!"这样的操作成功提示，据此可以设置检查内容为弹出框信息是否是"员工信息已保存!"的 GUI 检查点(如图 11-34 所示)。

"增加员工"操作完成后，单击 WinRunner 工具条中的停止按钮，WinRunner 自动结束测试脚本的录制，并将脚本显示在脚本编辑区。

② 增强测试脚本。录制完成的测试脚本只包含了我们在录制过程中输入的一组数据，使用这个脚本进行测试有很大的局限性，需要使用不同的数据执行该功能，以提高测试的充分性。因此，需要对上一步骤中得到的脚本做进一步处理，也就是对测试脚本中的新增员工信息进行参数化。参数化脚本的方法如下：在 WinRunner 的菜单中依次选择 Table→Data Table Wizard，按照数据表向导的提示单击 Next 按钮，选择要进行参数化的值设定相应的参数，本质上就是测试脚本执行期间的数据来源。如图 11-35 所示，为员工姓名设置参数为"员工姓名"，对应数据表(一个 Excel 文件)的"员工姓名"列，其他测试数

据的参数设置以此类推。

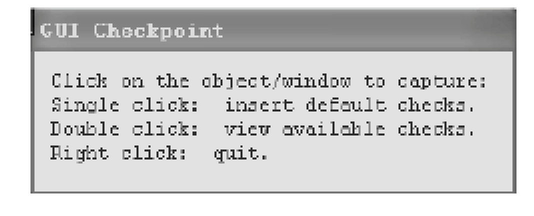




图 11-34 检查点



图 11-35 脚本参数化

参数化完成后，打开 Data Table，可以为员工信息准备测试数据，根据测试用例的设计输入正常的、非法的数据信息（如图 11-36 所示）。编辑数据表后，保存测试数据。至此，测试脚本即准备完毕，执行脚本即可得到有关该功能的自动测试结果（如图 11-37 所示）。

另外，WinRunner 提供了邮件传递测试结果的功能，如图 11-38 所示可以在 WinRunner（选择：Tools→General Options→Notific ations→E-mail）中设置测试结果接收人的 E-mail 服务器地址和 E-mail 地址，这样 WinRunner 在测试执行结束后可以按照邮件配置信息自动将测试结果发送到指定邮箱中。

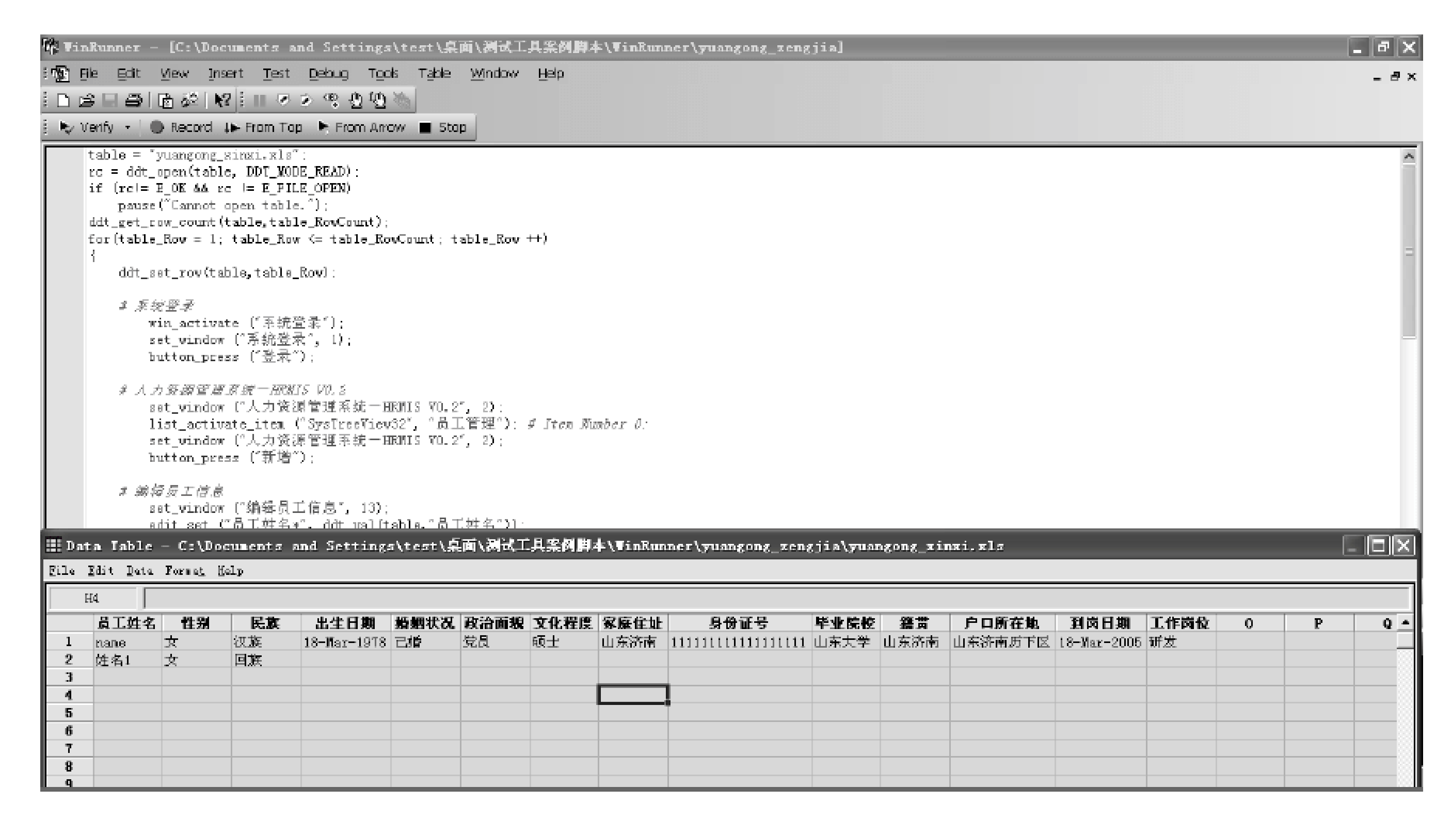


图 11-36　输入测试数据

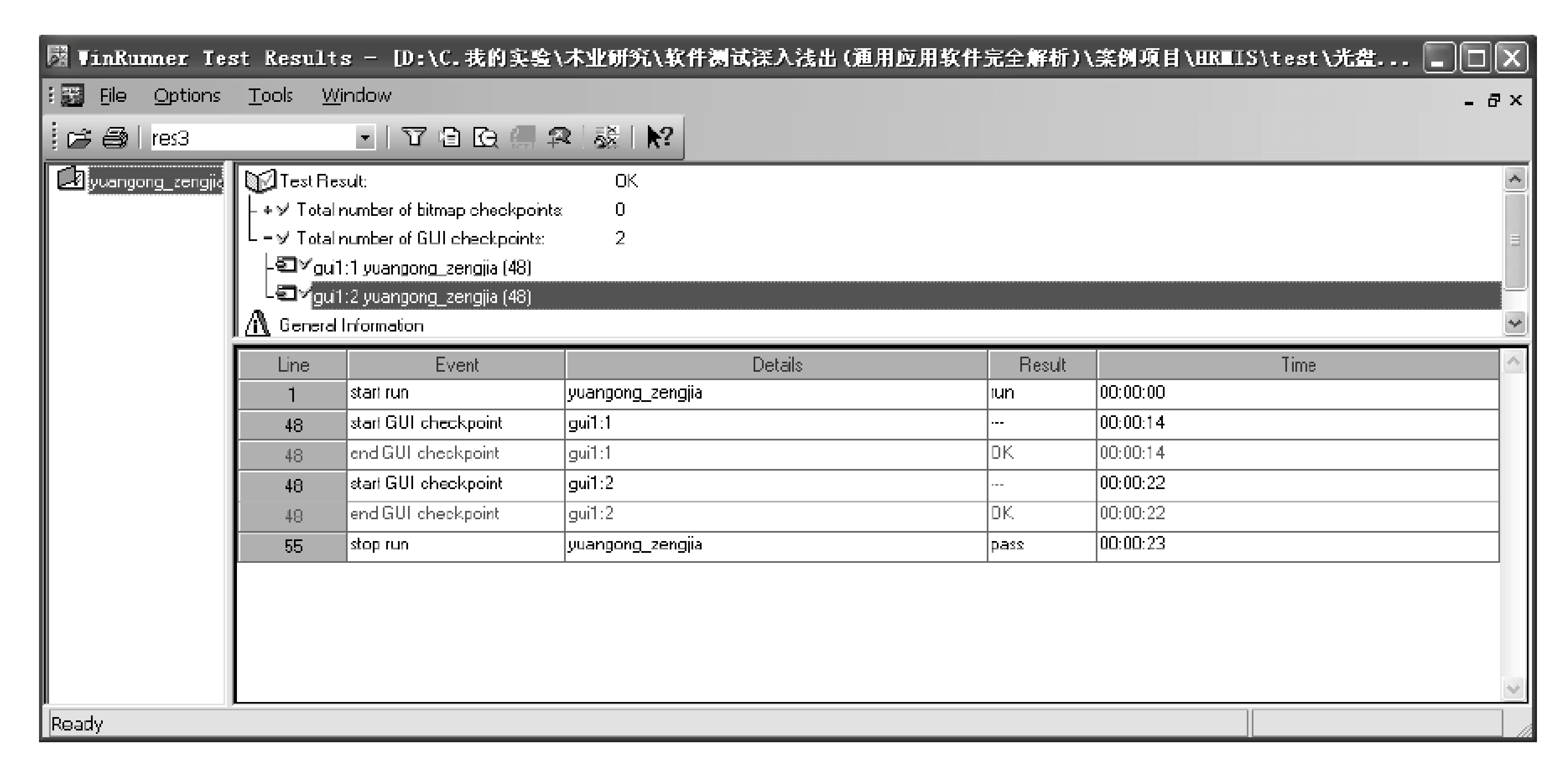


图 11-37　WinRunner 测试结果

在 WinRunner 运行过程中，如果运行结果与检查点不一致，WinRunner 会弹出窗口显示相应信息，这会导致测试中断，需要人工干预才能继续执行，这与我们所要求的无人值守的自动化测试执行是不一致的。为了避免这种情况的发生，可以在 WinRunner（选择 Tools→General Options→Run→Settings）中清除 Break when verification fails 选项，这样在测试执行过程中就不会因检查点不一致而导致测试中断了（如图 11-39 所示）

（2）应用自动化测试工具 QTP。

① 录制测试脚本。首先启动 HRMIS，双击"员工管理"，进入员工管理界面，然后启动 QTP，选择插件 Visual Basic，单击录制按钮，出现 Record and Run Settings 窗口（如

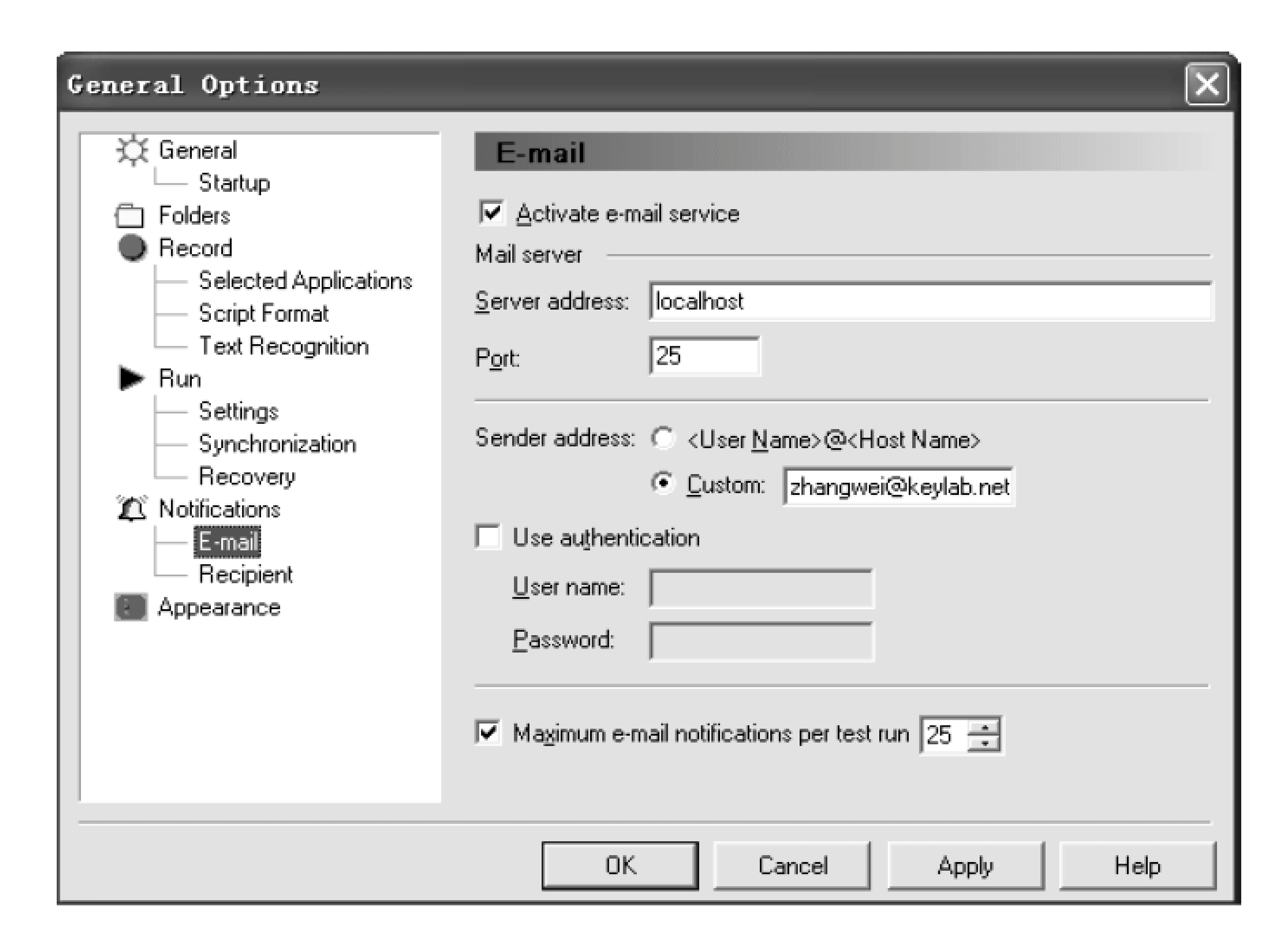


图 11-38　测试结果接收 E-mail 设置

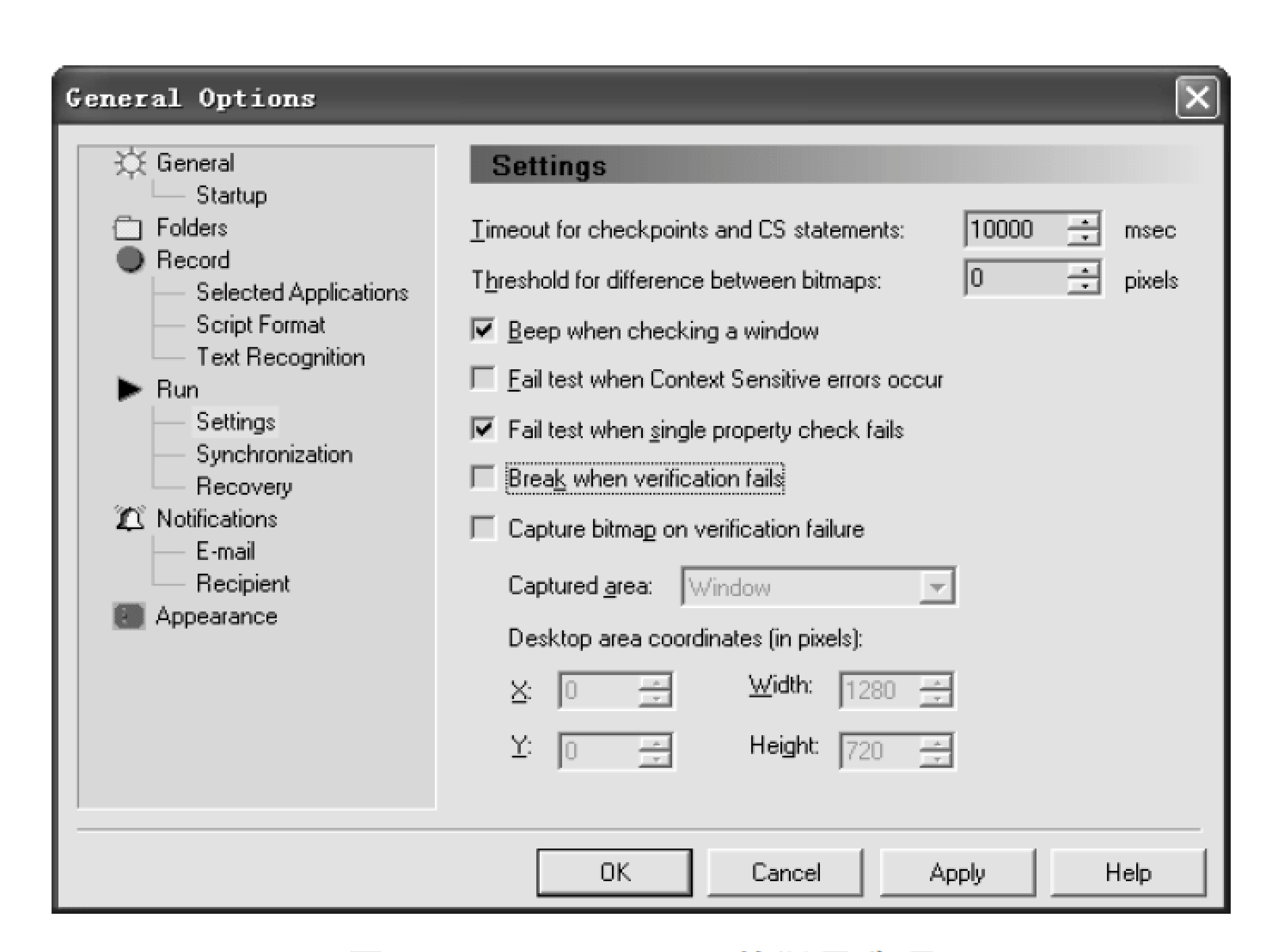


图 11-39　WinRunner 的通用选项

图 11-40 所示)。这里选择 Record and run test on any open Windows-based application，单击“确定”按钮，并切换到人力资源管理系统的员工管理界面。

应用 HRMIS 新增员工记录(见图 11-41)，QTP 会自动探测、记录该笔业务的操作过程和数据信息。操作完毕(操作至弹出员工增加信息成功窗口，单击“确定”按钮后该窗口关闭为止)，单击 QTP 的“停止”按钮，停止脚本录制，QTP 使用自己的语法规则自动生成和表示新增员工操作的业务仿真脚本(见图 11-42)。回放该脚本即可模拟 HRMIS 的一次“新增员工”操作。

② 增强测试脚本。对测试脚本中的新增员工信息进行参数化。在 QTP 的测试脚本

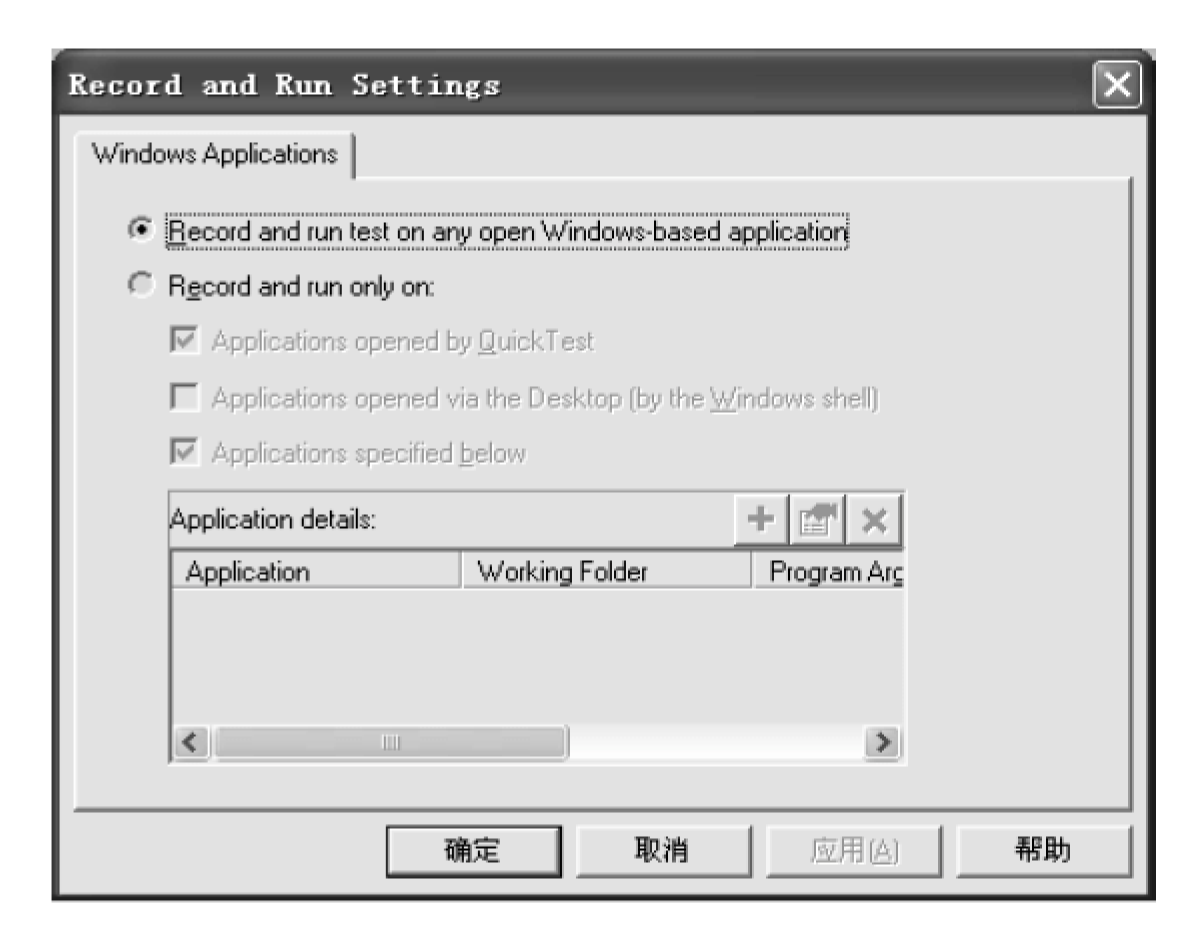


图 11-40 Record and Run Settings 窗口

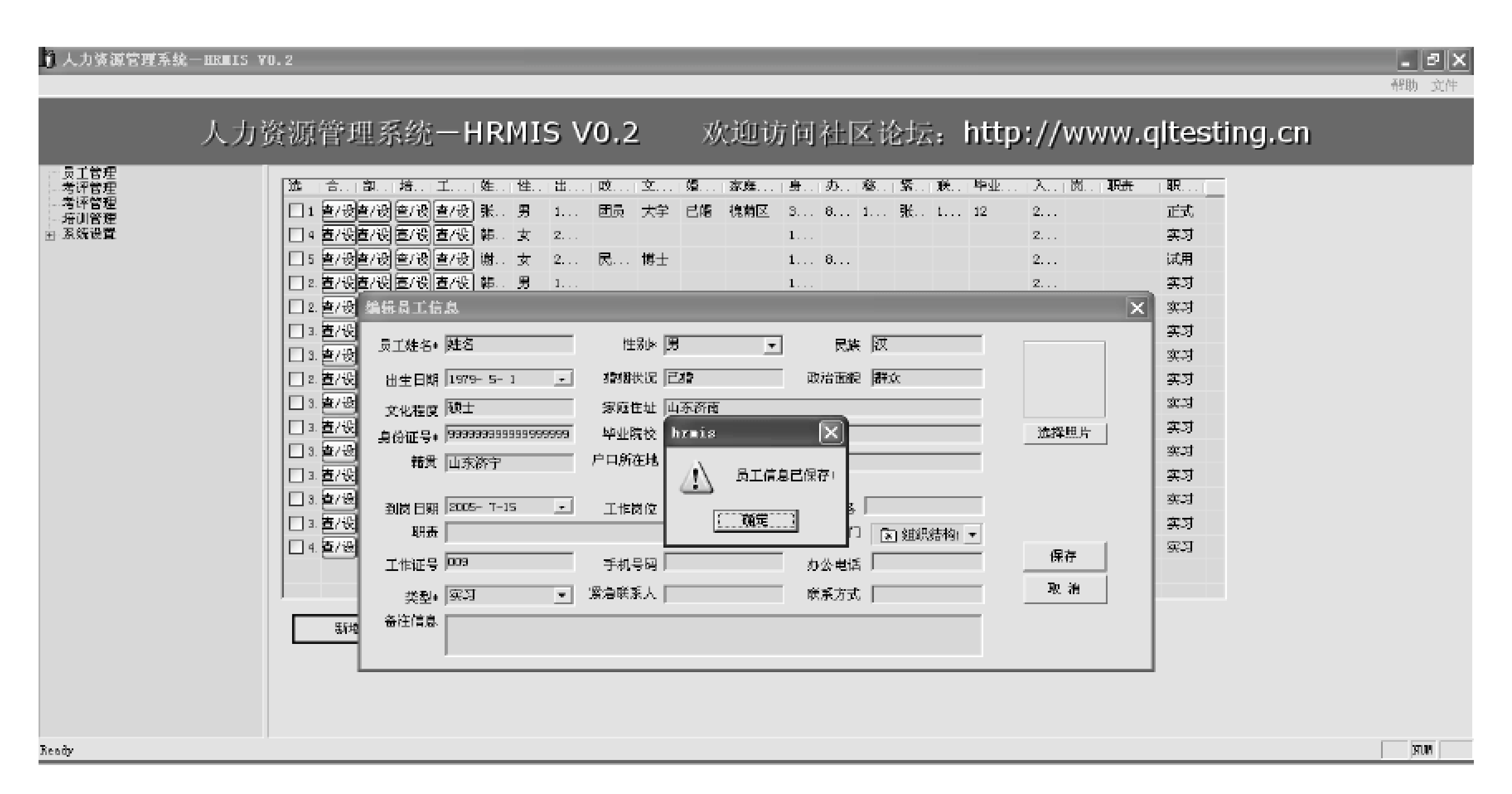


图 11-41 新增员工

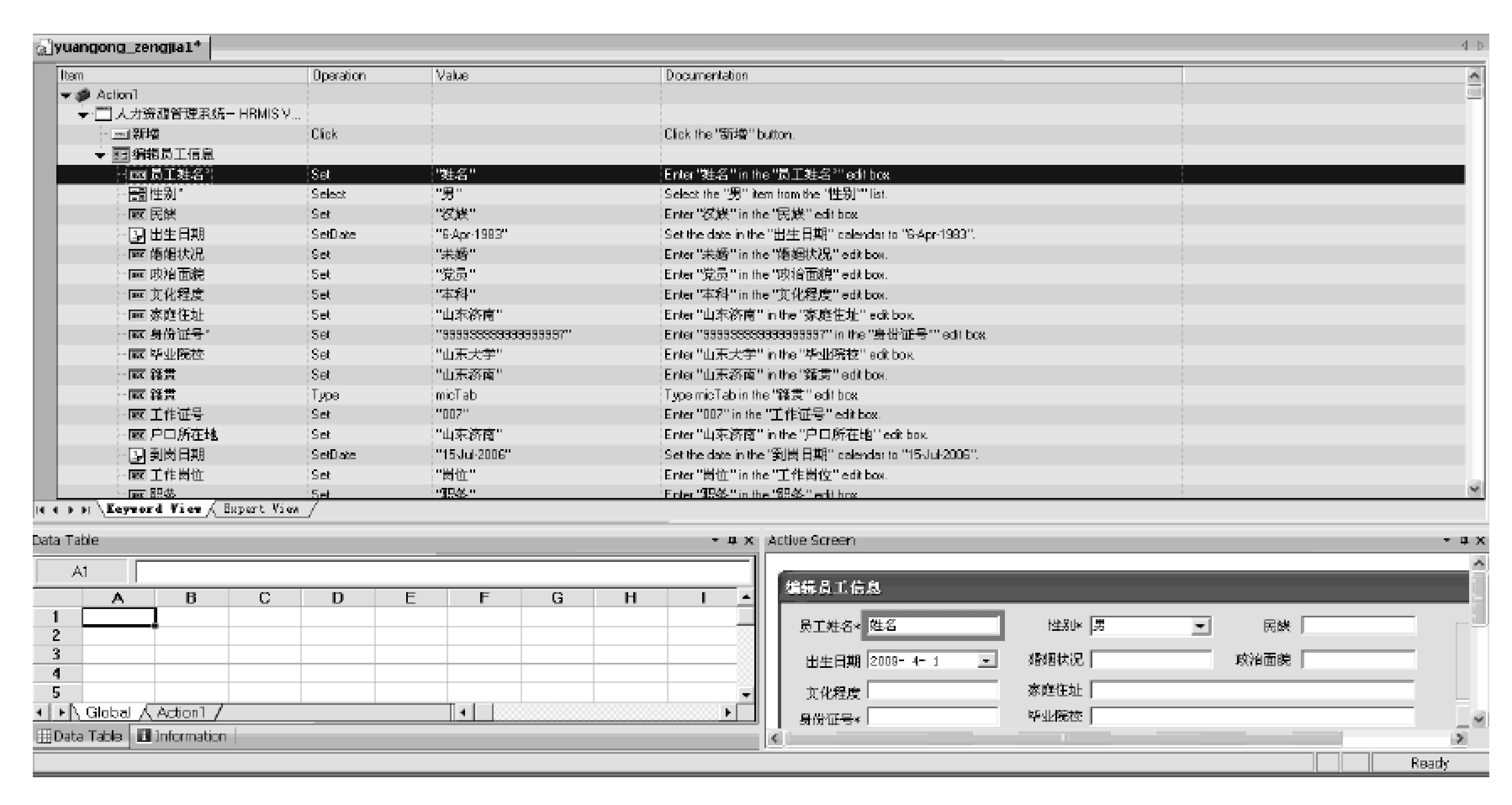


图 11-42 QTP 生成的测试脚本

中选择要进行参数化的字段，单击对应的 Value 右边的图标，进行参数化操作（如图 11-43 所示）。

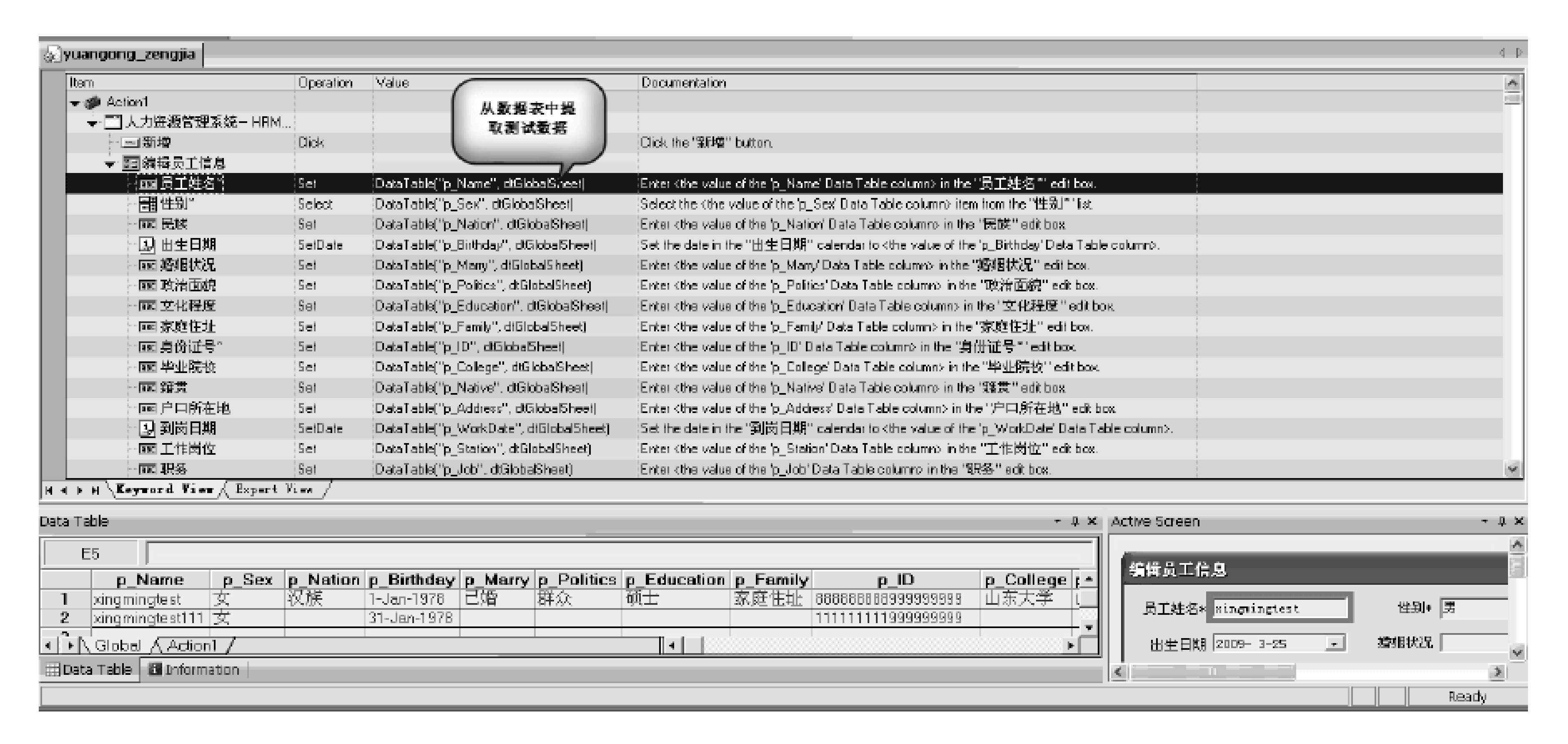


图 11-43　QTP 脚本参数化

③ 调试、运行测试脚本。通过 QTP 的 tools 菜单中的有关命令，对脚本进行相应的调试，单击菜单中的运行按钮，运行测试脚本，得到测试结果，对测试结果进行分析，得出测试结论（见图 11-44）。

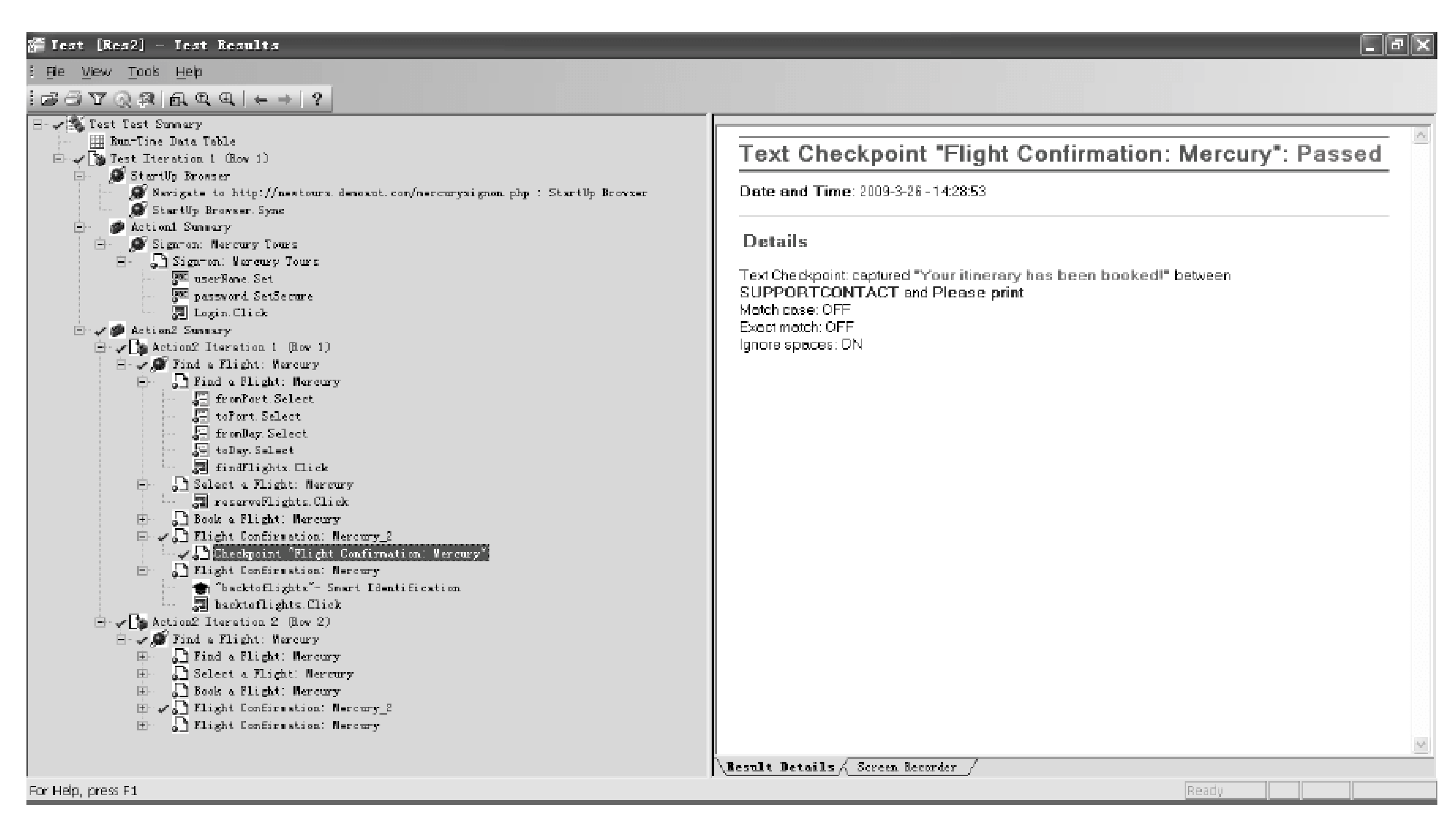


图 11-44　测试结果

这个结果按照“窗体—对象”的结构组织，完整显示了每次迭代的测试动作、测试所输入的数据和执行结果（Passed 或者 Failed）。在本例中，测试中未发现错误，全部测试结果为 Passed。可以进一步查看在脚本中设置的检查点，从得到的信息中可以看到所设置的 Text Checkpoint 检查点检查无误，执行结果为 Passed。

有关 WinRunner、QTP 自动测试工具的使用部分可以参考第 16 章功能测试工具的有关内容。

测试结论：

Text Checkpoint "员工信息已保存!": Passed

Date and Time: 2009-4-1 - 14:10:51

Details

Text Checkpoint: captured "员工信息已保存!"
Match case: OFF
Exact match: OFF
Ignore spaces: ON

【关于自动化测试执行的几点建议和忠告】

① 首轮测试尽可能减少自动化测试程度，因为自动化测试成本比较高，而且被测试系统处于不稳定期，功能经常发生变更，导致自动化测试脚本的维护工作量加大。

② 实践证明，自动化测试发现缺陷的效率比较低，自动化测试一般应用在比较稳定的系统的回归测试期间，用以进行功能确认。

③ 测试组织推动和发展自动化测试的主要动力多数来源于对减轻测试工作量的一个预期，但是实际上测试组织在引入自动化测试的初期往往并不能带来工作量的减少，反而使测试工作变得更加困难，导致不如手工测试效率高。这是因为，任何工具的引入都有一个学习和适应的过程，另外，自动化测试的适用性评估也很重要，例如产品级的测试就比项目级的测试更有必要引入自动化测试技术。

2. 性能测试执行

性能测试需要考查被测系统本身的响应时间、交易量等指标，往往需要模拟大的并发量，这种情况下很难采用人工的方式去进行。因此性能测试通常都是借助专业的性能测试工具完成，比如 LoadRunner(有关 LoadRunner 的使用在第 17 章有专门介绍)。

在本案的性能测试方案中，设计了两个测试场景，一个以系统登录为主要考查业务，另一个是信息修改的混合场景。系统登录是一个单一场景，操作起来比较简单，下面主要看一下信息修改这个混合场景(有关这个场景的描述请参考 10.3.4 节)的性能测试执行过程。

1) 录制(编制)测试脚本

混合场景中这两个业务的模拟操作脚本使用 LoadRunner 的虚拟用户生成器(Virtual User Generator，VUG)来开发，LoadRunner 提供了方便的 Record-Replay(录制回放)脚本开发方式，当然在足够了解 LoadRunner 的脚本语法之后，也可以像编程序一样来编写自己的测试脚本，那样可以更加灵活自主地满足自己的测试需求。在此，我们以录制模式来完成测试脚本的开发。

(1) 协议选择。LoadRunner 支持多种协议，在此我们根据 HRMIS 的架构和工作原理，我们可以确认使用 ODBC 协议来录制脚本(如图 11-45 所示)。注意，脚本录制协议如果不正确，可能会导致无法获取正确的脚本内容，而导致无法回放。

根据本案的测试场景要求(详细内容请参阅 10.3.4 节)，可以在脚本录制过程中定义并插入相应的事务(如图 11-46 所示)。

(2) 脚本结构及脚本增强。VUG 生成的脚本分为三部分：vuser_init(图 11-47 中标注 1)、vuser_end(图 11-47 中标注 3) 和 Action(图 11-47 中标注 2)。其中 vuser_init 和

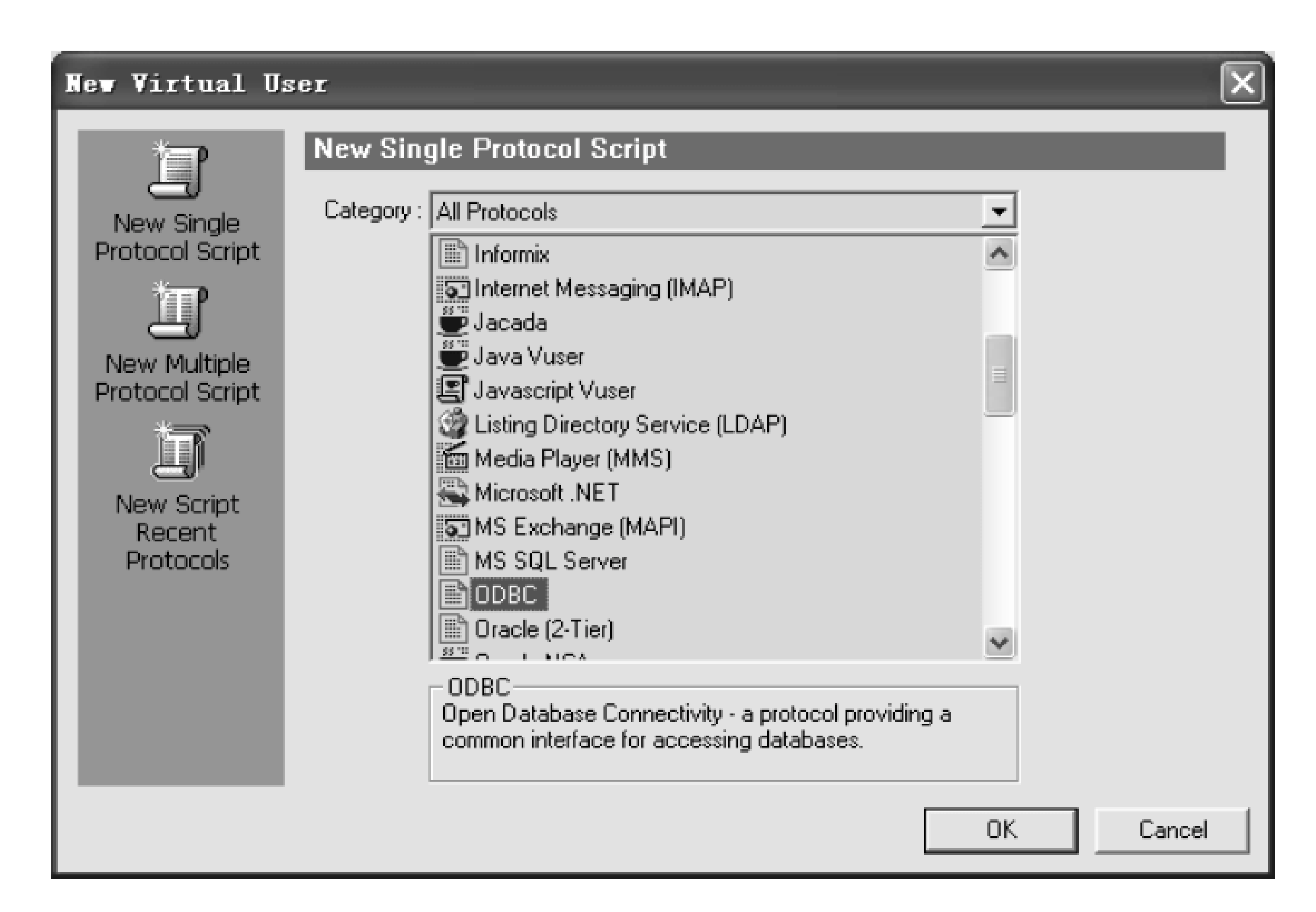


图 11-45　协议选择

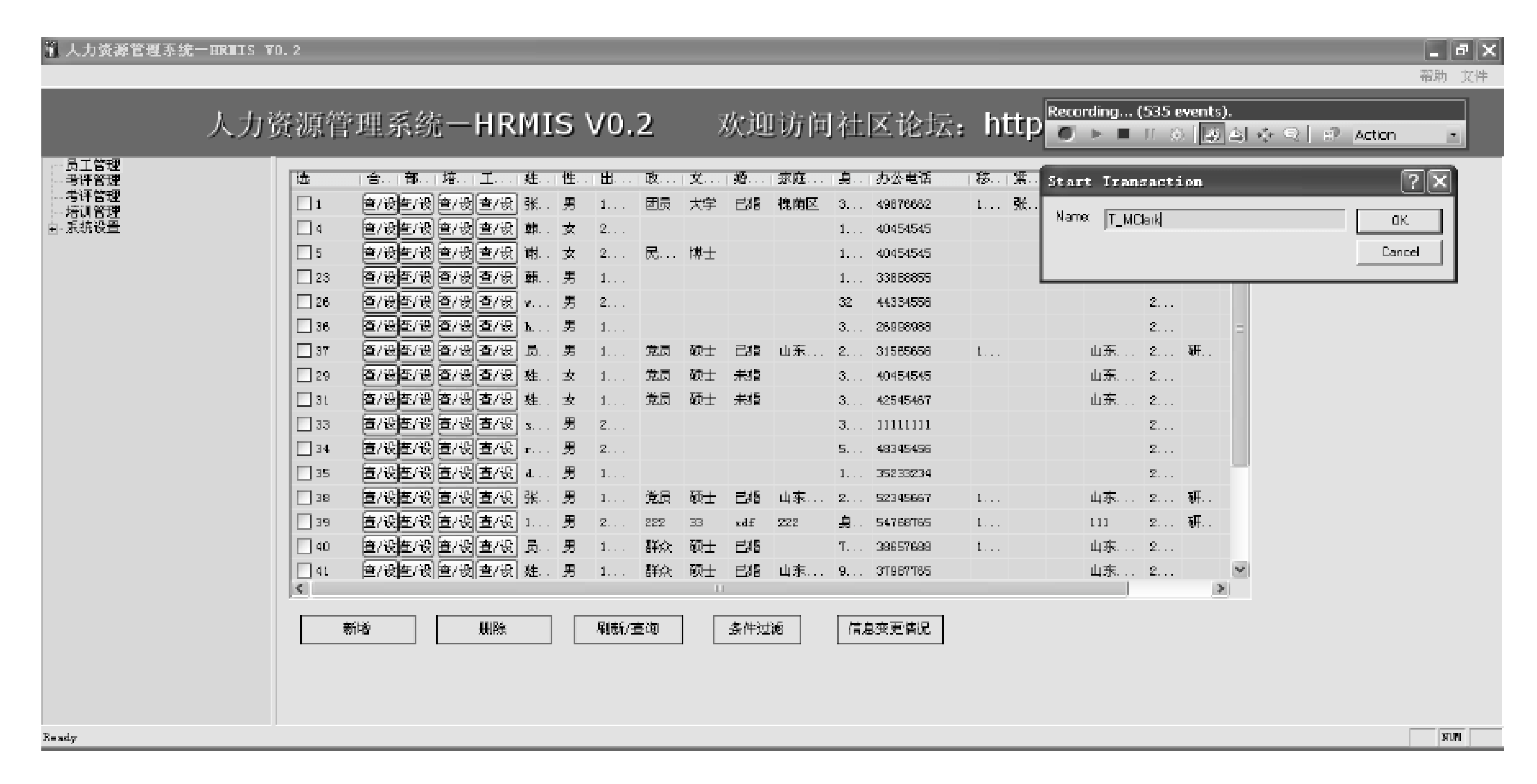


图 11-46　插入事务

Start Page | xiugai_dianhua - ODBC | xiugai_jigou - ODBC

vuser_init —— 1
Action —— 2
vuser_end —— 3

```
lr_rendezvous("R_hunhe");        —— 4
lr_start_transaction("T_MClerk");  —5

lrd_open_cursor(&Csr27, Con3, 0);
lrd_open_cursor(&Csr28, Con3, 0);
lrd_stmt(Csr28, "SqlType=SQL_ALL_TYPES;", -1, 3, 4 /*SQLGetTypeInfo*/, 0);
lrd_bind_cols(Csr28, BCInfo_D74, 0);
lrd_fetch(Csr28, -52, 1, 0, PrintRow8, 0);
GRID(8);

lrd_close_cursor(&Csr28, 0);
lrd_assign(&_1_D75, "44444444", 0, 0, 0);      //修改后的电话
lrd_bind_placeholder(Csr27, "1", &_1_D75, LRD_BIND_FOR_INPUT_ONLY, 0);
lrd_assign(&_2_D76, "{number}", 0, 0, 0);      //对应员工的序号
lrd_bind_placeholder(Csr27, "2", &_2_D76, LRD_BIND_FOR_INPUT_ONLY, 0);
lrd_assign(&_3_D77, "{Tel}", 0, 0, 0);      //修改前的电话
```

图 11-47　LoadRunner 脚本结构

vuser_end 在一个脚本中都只能存在一个，不能再分割，Action 在脚本中可以存在多个，要创建新的 Action 可以通过单击菜单 Action→Create New Action，新建一个 Action(见图 11-48)。

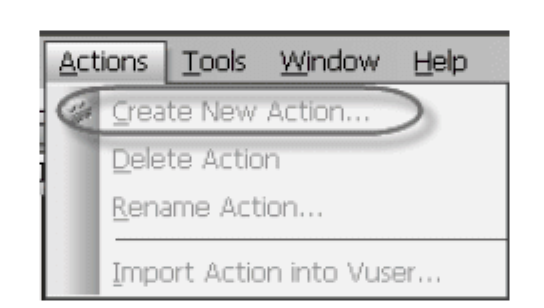


图 11-48 创建 Action

在重复执行测试脚本时，vuser_init 和 vuser_end 中的内容只会执行一次，重复执行的只是 Action 中的部分。因此，我们在录制脚本时如果需要录制登录部分且登录部分不是场景中的关键业务时，我们一般将登录部分放到 Vuser_init 中，把登录后的操作部分放到 Action 中，把注销退出系统的部分放到 Vuser_end 中。

录制的脚本只是一次业务操作的模拟，要仿真模拟实际应用环境还需要对脚本做进一步处理或称增强。首先，为了方便考查业务处理时间，我们要对脚本中的某项操作进行事务定义，这样 LoadRunner 就能够识别到这个事务，能够采集其相应的指标信息。因此，根据测试场景要求，我们在“员工信息修改”这个脚本中，使用 lr_start_transaction("T_MClerk")定义标识为 T_MClerk 的一个事务(图 11-47 中标注 5)，在“培训机构修改”脚本中我们使用 lr_start_transaction("T_MTrain")定义一个标识为 T_MTrain 的事务。

在 LoadRunner 中支持并发情况模拟的是集合点机制，为了模拟用户并发情况，可以在脚本中增加一个集合点定义，集合点定义使用语句 lr_rendezvous("R_hunhe")(图 11-47 中标注 4)。注意集合点的定义要在事务定义之前。

除了事务和集合点定义之外，还应该使测试数据多样化，在 LoadRunner 中使用参数化这一方法满足这一要求。选择什么样的数据进行参数化，完全决定于测试要求和系统数据处理流程。在本案中，脚本“员工信息修改”主要参数化两个数据，一个是员工记录编号，一个是办公电话(如下脚本代码中黄色标识区)。

```
Action()
{
    ⋮
    lr_rendezvous("R_hunhe");
    lr_start_transaction("T_MClerk");

    lrd_close_cursor(&Csr28, 0);
    lrd_assign(&_1_D75, "44444444", 0, 0, 0);          //修改后的电话
    lrd_bind_placeholder(Csr27, "1", &_1_D75, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_2_D76, "{number}", 0, 0, 0);          //对应员工的序号
    lrd_bind_placeholder(Csr27, "2", &_2_D76, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_3_D77, "{Tel}", 0, 0, 0);             //修改前的电话
    lrd_bind_placeholder(Csr27, "3", &_3_D77, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_4_D78, "-858993460", 0, 0, 0);
    lrd_bind_placeholder(Csr27, "4", &_4_D78, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_stmt(Csr27, "UPDATE 'hrmis'.'T_YG' SET 'BGDH'=? WHERE 'YGBH'=? AND 'BGDH'=?
    AND "
        "'BMBH'=?", -1, 1, 0 /* None */, 0);
```

```
    ⋮
    return 0;
}
```

由于办公电话可以是任意符合要求的数字串，因此办公电话和员工编号之间没有约束关系，这两个可以独立准备测试数据（如图 11-49 是参数 Tel 的测试数据）。

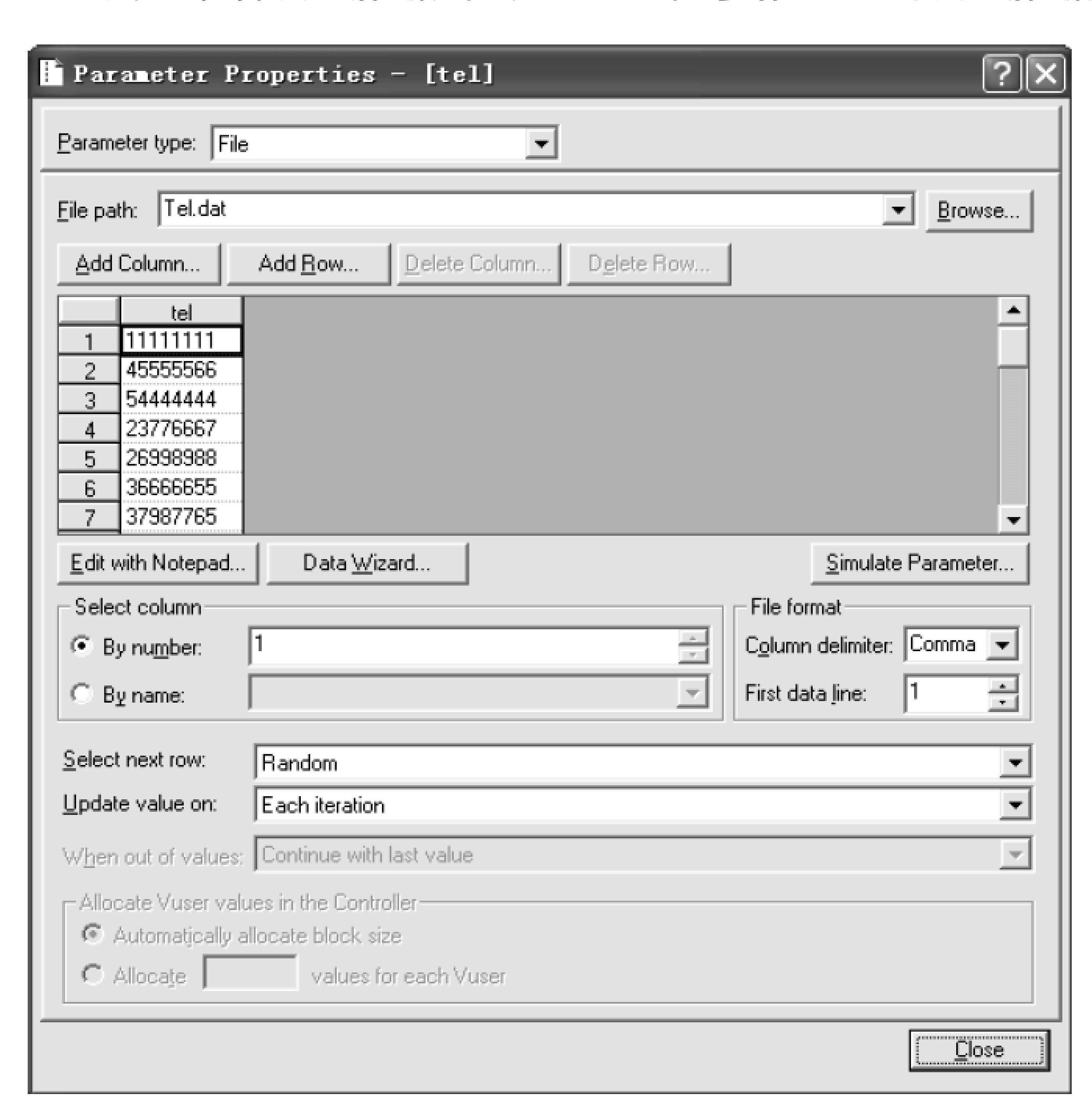


图 11-49　Tel 的测试数据

同样原理，以下是"培训机构修改"的参数化后的测试脚本及其机构名称的数据（如图 11-50 所示）。

```
Action()
{
    ⋮
    lrd_close_cursor(&Csr11, 0);
    lrd_assign(&_1_D66, "{jigouming_new}", 0, 0, 0);            //修改后的机构名称
    lrd_bind_placeholder(Csr10, "1", &_1_D66, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_2_D67, "{dizhi_new}", 0, 0, 0);                //修改后的机构地址
    lrd_bind_placeholder(Csr10, "2", &_2_D67, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_3_D68, "{lianxiren_new}", 0, 0, 0);            //修改后的联系人
    lrd_bind_placeholder(Csr10, "3", &_3_D68, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_4_D69, "{lianxifangshi_new}", 0, 0, 0);        //修改后的联系方式
```

```
    lrd_bind_placeholder(Csr10, "4", &_4_D69, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_5_D70, "{email_new}", 0, 0, 0);              //修改后的电子邮件
    lrd_bind_placeholder(Csr10, "5", &_5_D70, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_6_D71, "{jigouhao}", 0, 0, 0);               //修改机构对应的序号
    lrd_bind_placeholder(Csr10, "6", &_6_D71, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_7_D72, "{jigouming_old}", 0, 0, 0);          //修改前的机构名称
    lrd_bind_placeholder(Csr10, "7", &_7_D72, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_8_D73, "{dizhi_old}", 0, 0, 0);              //修改前的机构地址
    lrd_bind_placeholder(Csr10, "8", &_8_D73, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_9_D74, "{lianxiren_old}", 0, 0, 0);          //修改前的联系人
    lrd_bind_placeholder(Csr10, "9", &_9_D74, LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_10_D75, "{lianxifangshi_old}", 0, 0, 0);     //修改前的联系方式
    lrd_bind_placeholder(Csr10, "10", &_10_D75,
        LRD_BIND_FOR_INPUT_ONLY, 0);
    lrd_assign(&_11_D76, "{email_old}", 0, 0, 0);             //修改前的电子邮件
    lrd_bind_placeholder(Csr10, "11", &_11_D76,
        LRD_BIND_FOR_INPUT_ONLY, 0);
    ⋮
    return 0;
}
```

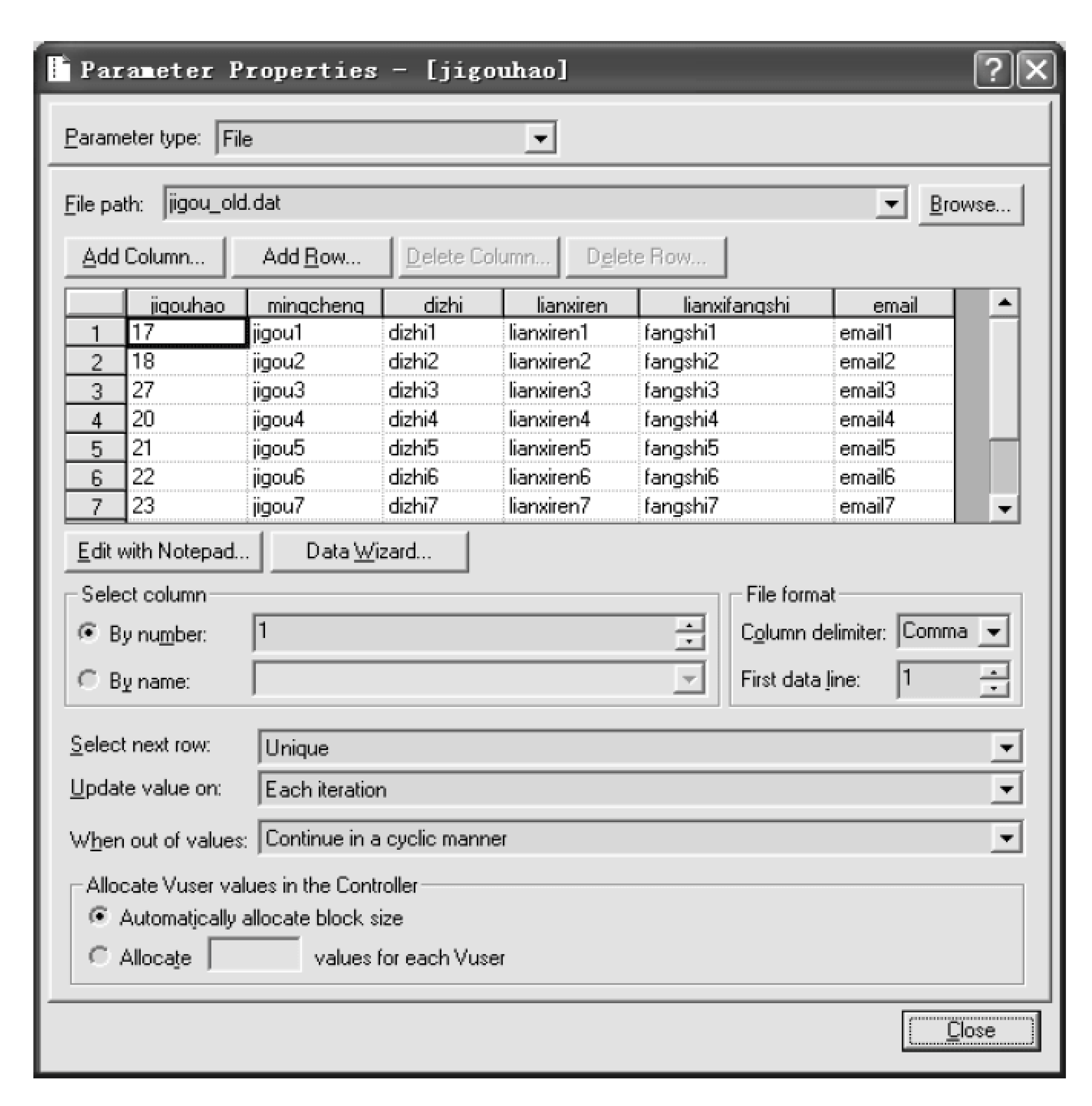


图 11-50　修改培训机构信息测试脚本用测试数据

2）场景设置

使用 LoadRunner 的 Controller 可以创建并运行测试，可以准确地模拟被测系统的预定工作场景。

我们要做的是一个包含员工信息修改和培训信息修改的两个脚本的混合场景，因此在创建场景时首先要把这两个脚本引入（如图 11-51 所示）。在 Scenario Schedule（场景规划）中，做如表 11-12 所示的设置。

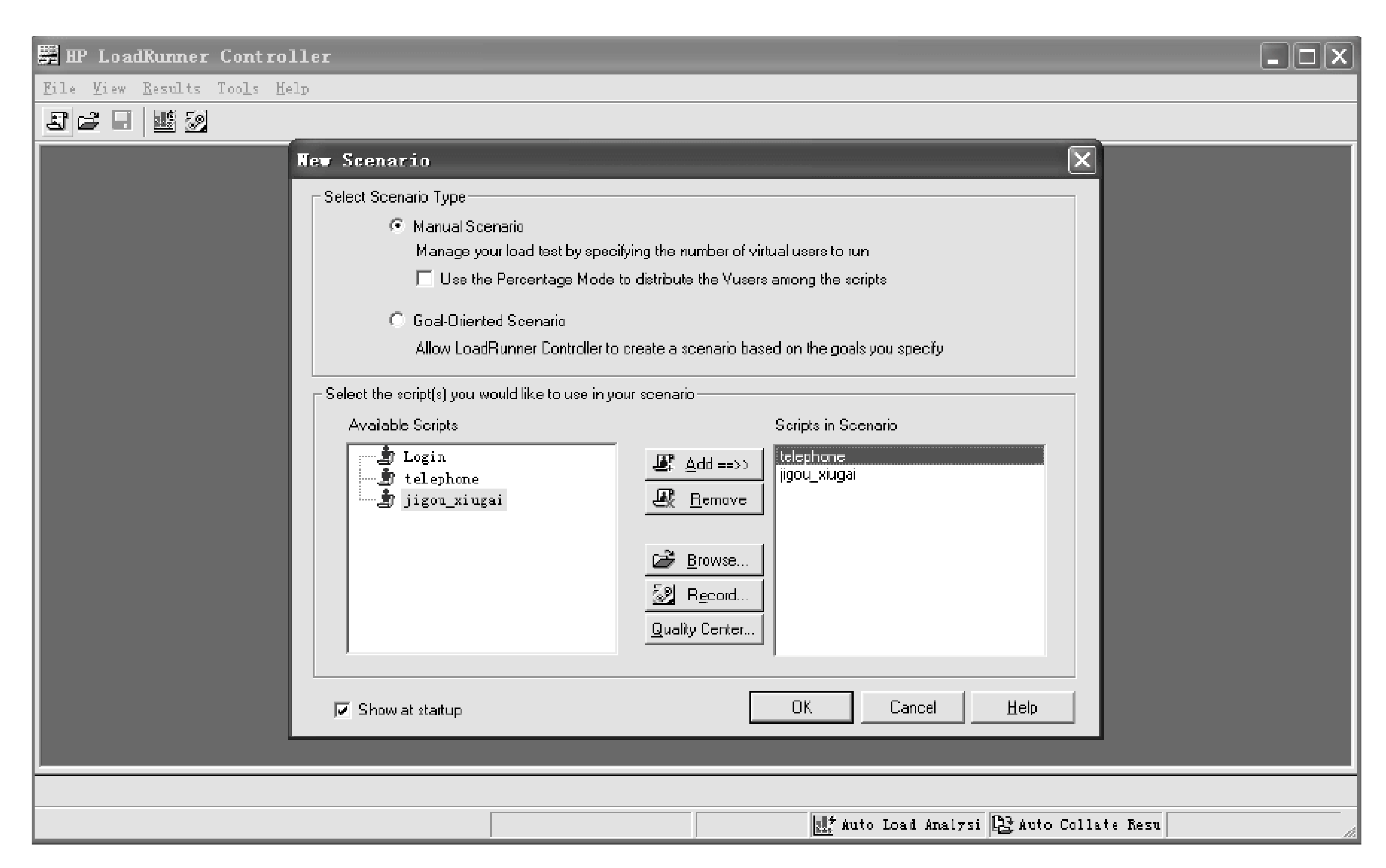


图 11-51　LoadRunner 场景设置

表 11-12　场景设置

选项	设　　置	解　　释
Initialize	Initialize each Vuser just before it runs	在虚拟用户运行前初始化
Start Vusers	start 20 Vusers:5 every 00:00:01 (HH:MM:SS)	每秒种启动 5 个用户，直至 20 个
Duration	Run for 00:05:00 (HH:MM:SS)	持续运行 5 分钟
Stop Vusers	stop all Vusers:5 ervey 00:00:30 (HH:MM:SS)	每 30 秒停止 5 个用户

LoadRunner 会将以上设置以图形的形式直观地显示出来（见图 11-52 右侧）。

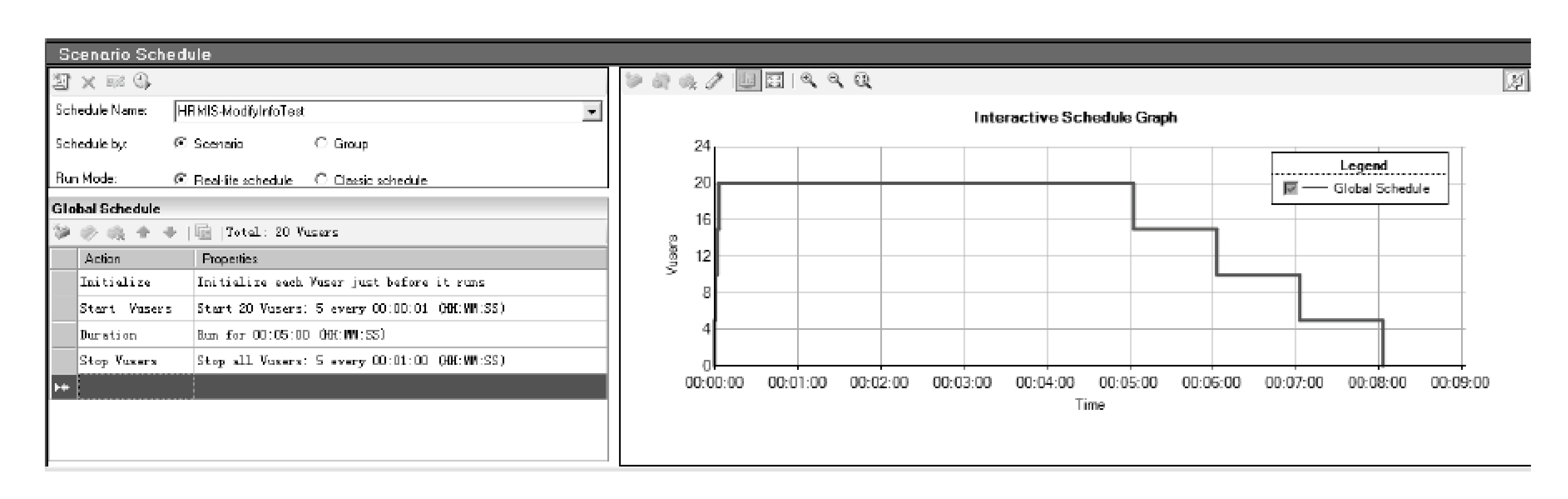


图 11-52　场景设置

在 Ctroller 中，我们可以对脚本的运行时参数进行设置(RTS)：在本案中我们设置 action 的迭代次数(pacing 选项 Number of Iterations)为 3，Think Time(思考时间)为 Ingore，忽略思考时间可以加大运行时系统处理压力(如图 11-53 所示)。

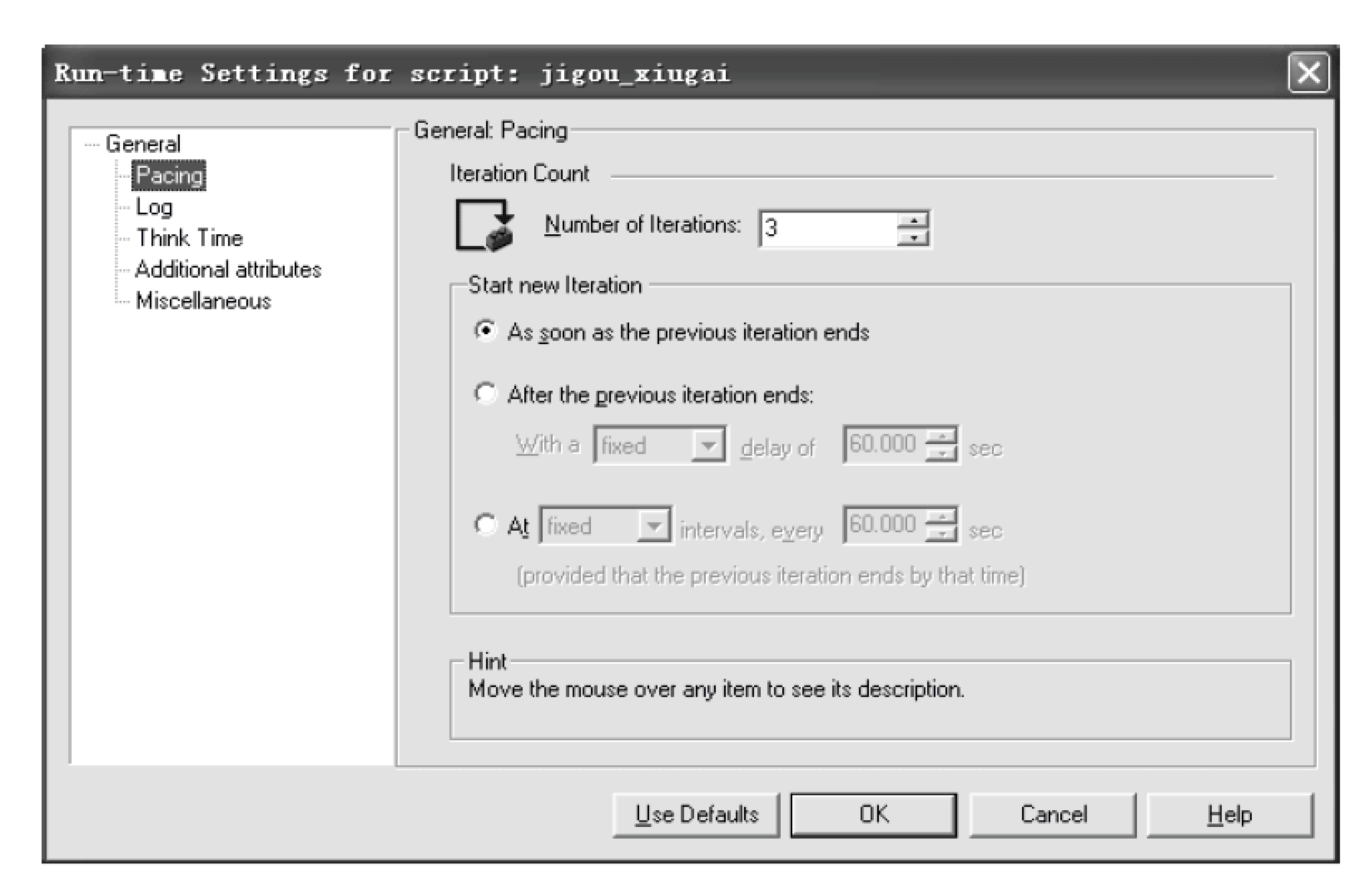


图 11-53　RTS

虽然在修改员工电话和修改培训机构信息的两个测试脚本中都加入了相同的集合点 R_hunhe，但是要实现多个用户同时进行修改员工电话或修改培训机构信息操作，还需要在 Controller 中设置集合策略。在 Controller 菜单中找到 Scenario，单击 Rendezvous，进入集合点信息窗口，选择集合点 R_hunhe，单击 Policy 按钮，进入集合策略制定窗口(如图 11-54 所示)。这里选择"Realease when 100% of all Vusers arrive at the rendezvous"，即所有的用户到达集合点后再继续执行后续的操作请求。

3) 测试结果分析

LoadRunner Analysis 提供了自动采集测试数据和分析结果的功能，通过分析以查找系统的性能故障，然后确定这些故障的根源。LoadRunner Analysis 可以回答以下问题：

(1) 是否满足了测试的预期目标？被测系统在高负载下，终端用户的事务响应时间、平均响应时间是多少？

(2) 系统的哪些部分导致性能下降？该网络和服务器的响应时间是多少？

(3) 通过将事务时间和后端监控器关联是否可以找到潜在性能故障？

对于本案例的测试，客户所关心的问题是 20 个用户并发下修改类交易的平均响应时间有没有超过 15s，对应到这个测试场景，则是 T_MClerk 和 T_MTrain 这两个交易事务。图 11-55 所示是 LoadRunner Analysis 生成的有关这两个交易的测试情况，从图中可以非常直观地看到：T_MClerk 在 20 个用户负载下，其最高交易平均响应时间是 0.857s，T_MTrain 在 20 个用户负载下，其最高交易平均响应时间是 0.093s，均远远小于预期的 15s。

为了进一步验证，我们继续深入查看一下 T_MClerk 和 T_MTrain 的负载下的响应

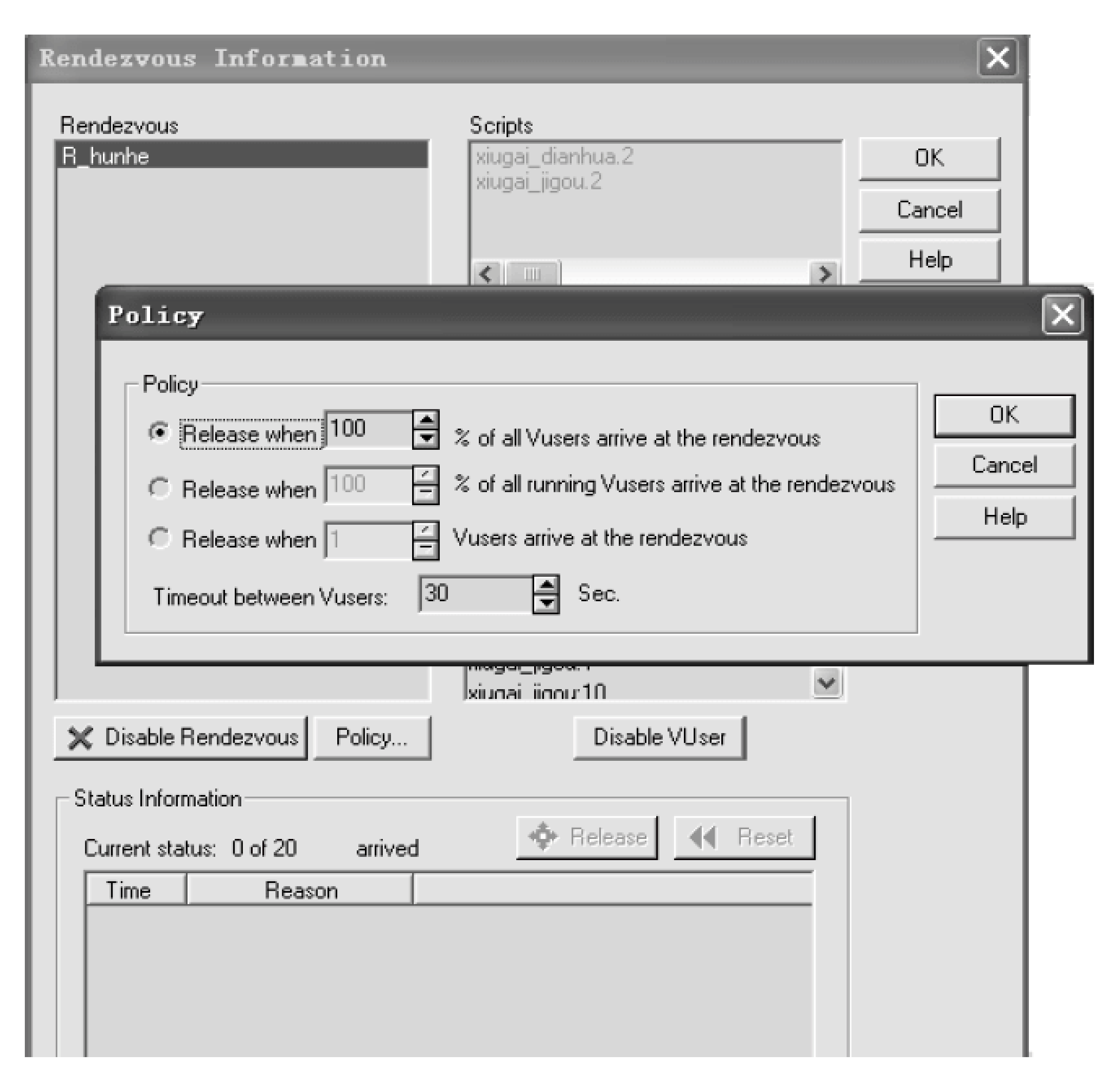


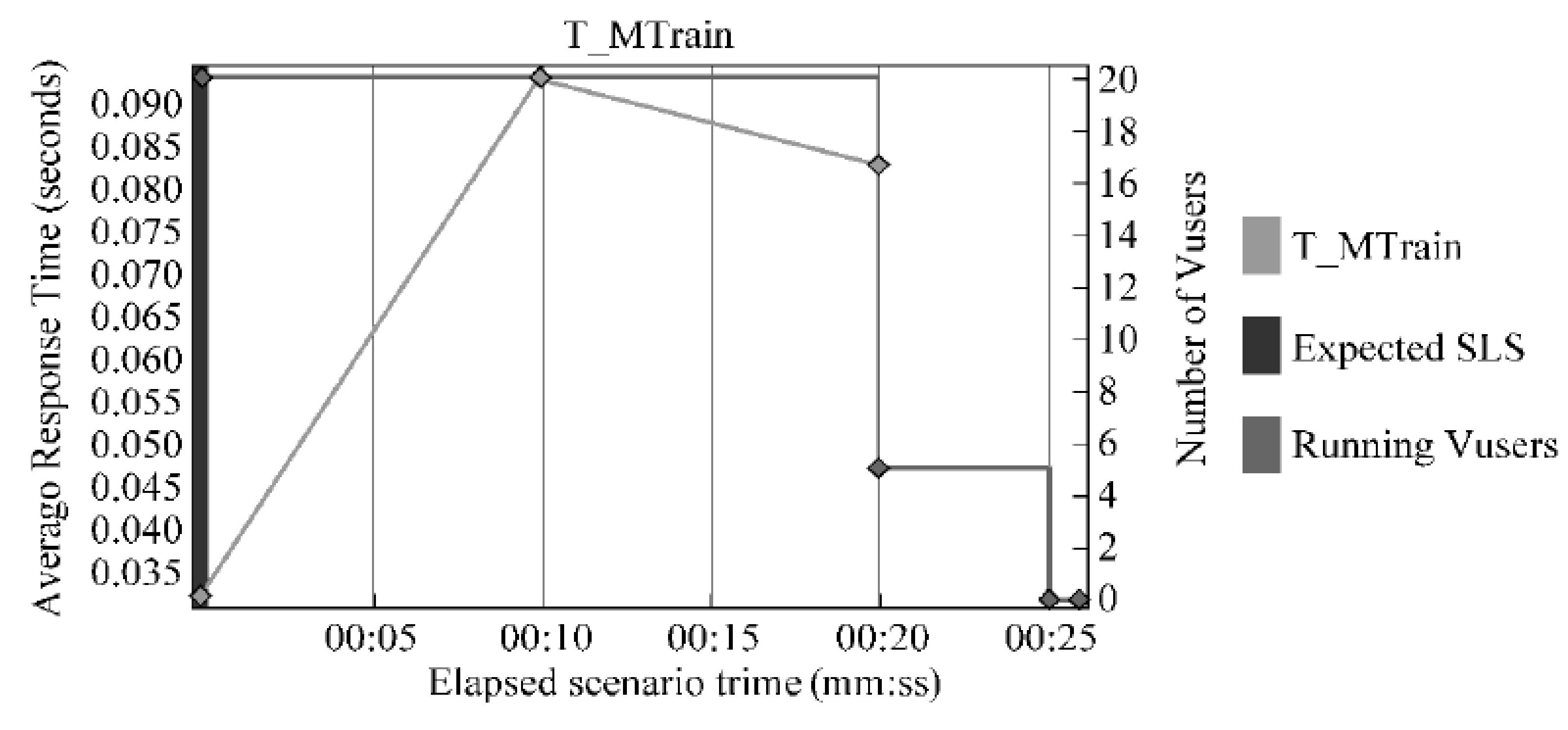
图 11-54 集合点策略

时间统计情况(如图 11-56 所示,该图来自 Load Runner Analysis 软件)。这个图给出了 T_MClerk 和 T_MTrain 这两个事务的响应时间详细变化情况,包含最小值、最大值和平均值。从图中可以看到 T_MClerk 和 T_MTrain 在整个场景运行期间其最大事务响应时间分别是 0.943s 和 0.319s,都远小于预期的 15s。

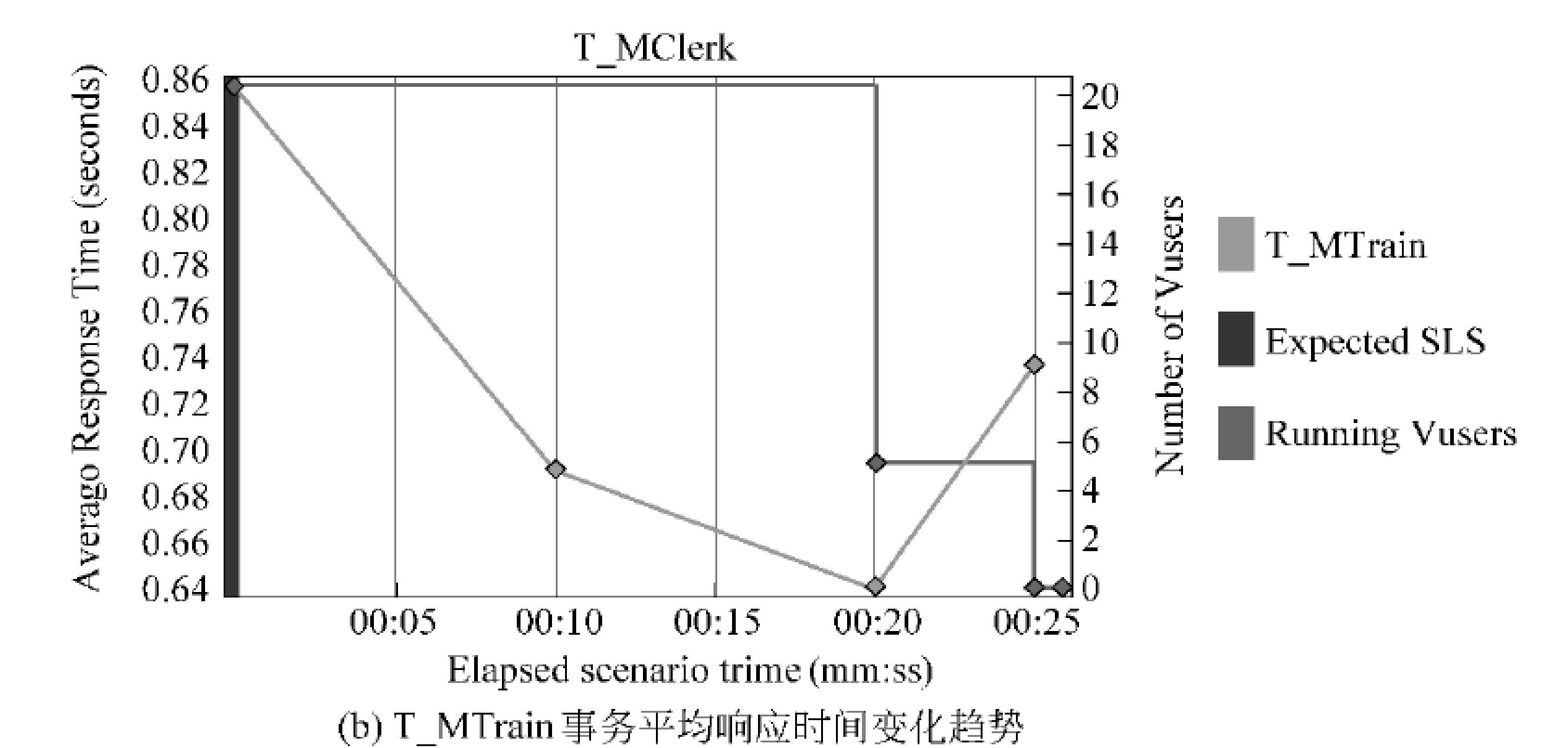
系统性能测试是一个复杂的过程,性能测试不仅仅要观察系统本身,同时对于它所在的环境也要有相应的观察,多方面结合起来才能够对系统的性能做一个接近于真实的评估。在本案例中,我们除了做上述必要的事务考查之外,还应对系统所在主机的资源使用情况进行考查。如图 11-57 所示是使用 LoadRunner 监视器获得的 HRMIS 的数据库服务器的资源使用情况,对照我们的场景设置和要求,从中可以判断出其四项资源使用指标也是在我们的预期范围之内的。

通过多方面的认真考查,我们基本上可以对系统的交易性能做出判断和确认,系统预期的性能指标 20 个并发用户下平均交易响应时间小于 15s 是完全可以满足的。(本章对于性能测试的内容基本上都是直接加以使用,有关性能测试部分的详细操作介绍可以参考第 10 章 LoadRunner 部分的内容。)

另外必须知道,性能测试通常情况下不是单纯的一次性能指标考察,进行性能测试更多的是为了对系统容量、系统用户体验等做更好的评估。因此,性能测试执行期间往往还伴随着大量的系统优化的工作,通过不断的性能测试,找出系统参数的最优化配置。实际上,我们上面所给出的 HRMIS 的性能测试结果,也是在系统多次优化、反复测试后,所取得的相对比较理想的一组性能指标。



(a) T_Clerk 事务平均响应时间变化趋势



(b) T_MTrain 事务平均响应时间变化趋势

图 11-55　事务分析

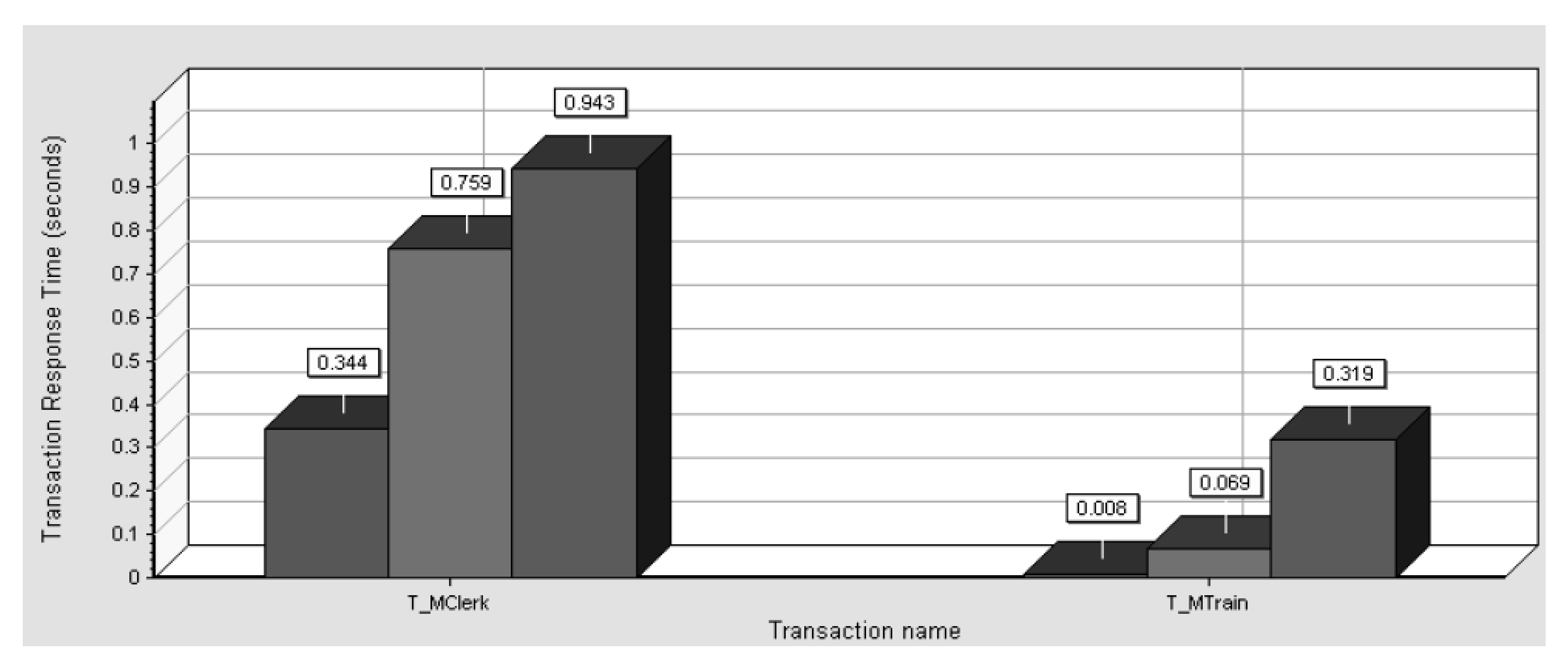


图 11-56　事务响应时间统计

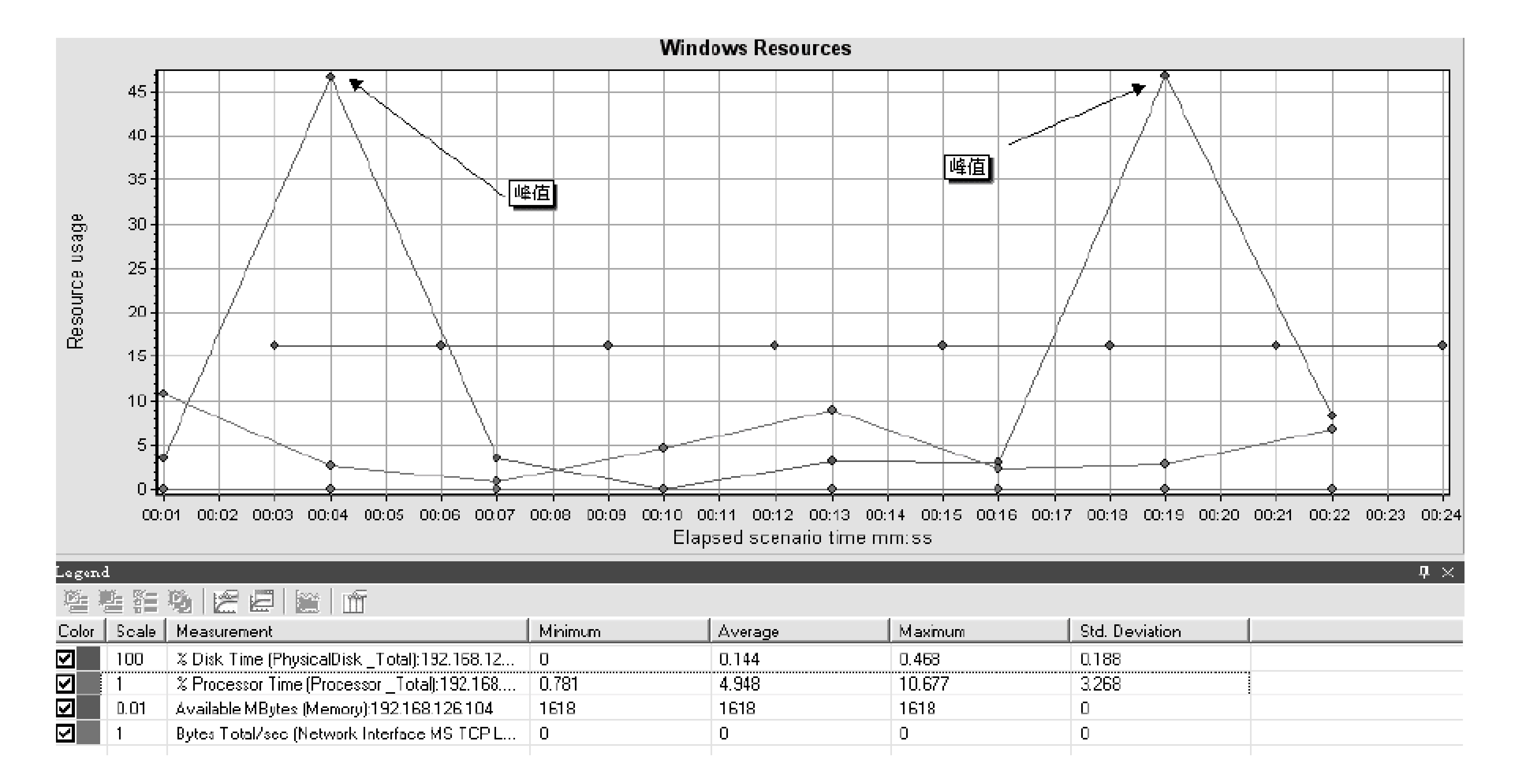


图 11-57　系统资源监控结果

本章小结

本章承接测试方法与测试设计部分，对软件测试执行过程的各项工作以及有关技术做了深入讲解，如测试准备、测试执行、缺陷管理、回归测试等内容。同时为了增强实用性，本章仍然秉承理论与实践结合的叙述形式，站在实践的角度对前一章所做的用例设计的执行过程进行了示范，并结合各类业界闻名的测试工具着重讲述了软件测试领域的难点单元测试、性能测试的执行和结果采集分析过程。

CHAPTER

第 12 章

测试评估

严谨的测试过程最终是为了获取一份严谨可信的测试报告，所谓“为山九仞，功亏一篑”，因此对于软件测试最后阶段工作的开展仍然马虎不得，绝不能因为最后“一筐土”而丢掉“九仞之山”。

测试过程的最后一步是分析测试结果，确定应用软件是否已成功或需要再次的测试，这就是测试评估阶段。测试评估要回答以下问题：

- 哪些经过了测试？
- 哪些通过了测试？
- 工作进度及效率如何？
- 发现了哪些缺陷？
- 哪些缺陷已经得到修改？
- 遗留缺陷对软件有何影响？
- 综合评价被测程序质量是否合格？

12.1 测试评估工作模型

测试评估本质上就是要判断测试是否已经充分，已经可以停止，可以出具测试报告了。测试评估要对整个软件测试实施过程进行复查，通过对测试过程关键数据的抽查和评价，来判断测试的充分性，如果含有风险较高的未测试项或者测试不充分项，则要决策是否需要进一步测试。从这个角度来说，测试评估是站在过程决策角度的一种度量，以度量数据为基础来干预测试进程。有关软件测试的度量还有基于过程改进目的的度量，这部分内容我们将在本书第四篇予以讨论。

把测试评估放到度量这样一个层面上来看，测试评估的工作模型可以抽象为以下内容（如图 12-1 所示）：

（1）选择确定所需要的度量指标，部分指标可能已经在测试计划中出现。从测试数据中抽取所需要的测试数据，计算有关指标的结果。

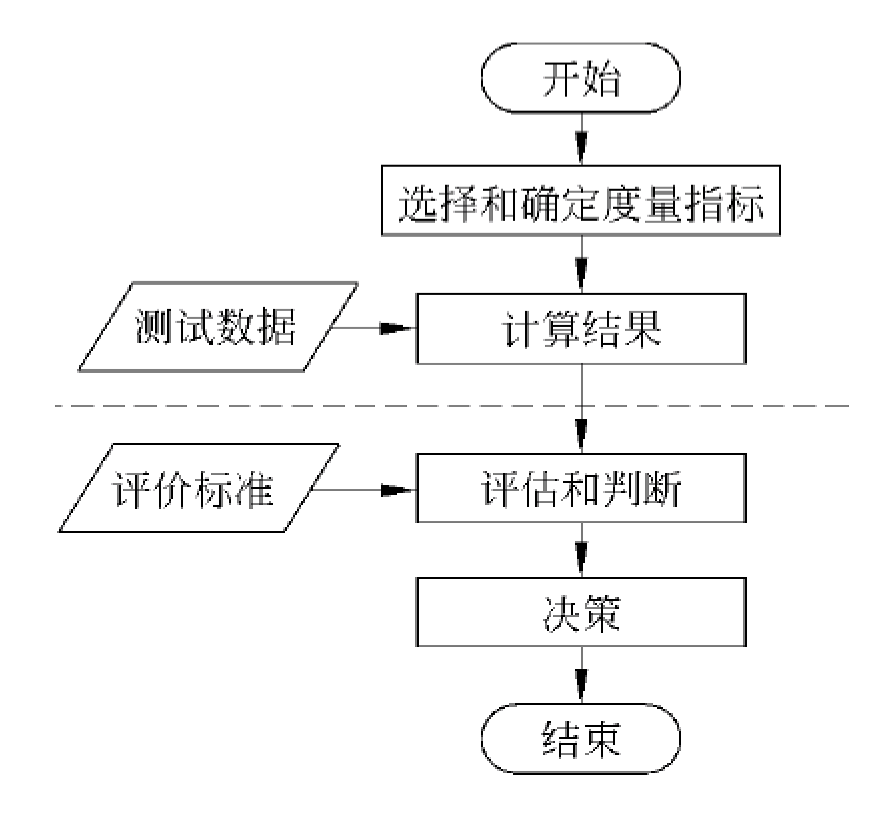


图 12-1 评估工作模型

(2) 根据通用的或者项目特例判断所得到的指标结果是否满足要求,如果满足要求则可以终止测试,编写测试分析报告,否则要重启一轮测试或者对测试不足部分补充测试,或者更改评价标准,但是这要经过所有管理人员的共同决策,并评估改变标准后的可能风险。

12.2 测试评估内容

一个充分的测试评估应该是站在全局的角度对已进行的和计划进行的测试活动进行关键数据采样,通过严谨地计算、判断以及从业经验等综合决策的过程。有关测试评估的研究是软件测试领域的一个热点,我们在本章介绍的内容也是基于最佳实践。

测试评估既然是全过程的,那么以测试生命周期为线索可以使评估全面、充分。其中测试计划阶段,我们主要评估资源配备情况、测试范围符合情况、计划任务完成情况、目标达成情况、风险度量和未测试项影响分析,其中以计划任务完成情况和未测试项影响分析尤为重要;测试需求阶段以需求覆盖情况、质量特性检测情况和填写需求跟踪矩阵为主要评估方向,这里面填写出需求跟踪矩阵是需求阶段评估的基础;测试方法与设计主要评估用例完整性、脚本完整性、脚本运行情况、用例执行情况和填写需求跟踪矩阵,用例执行情况、脚本运行情况和填写需求跟踪矩阵是这几个方面的重点;测试执行阶段以测试周期考查、MTBF 和缺陷分析这三个方面为主要评估方向,其中缺陷分析是测试执行阶段评估的核心内容(如图 12-2 所示)。

12.2.1 测试计划中的评估

测试计划对于测试实施具有指导作用,对计划的评估不是评审,也就是说不是评价质量的好坏,而是对计划的实施情况进行考查。测试计划的评估是判断“说要做的是否已经做到了”。有关测试计划评估可以从以下几个方面着手:

1. 资源配备情况

资源是否充分对测试质量有直接影响,因此,我们在评估时要对资源配置及其中间的

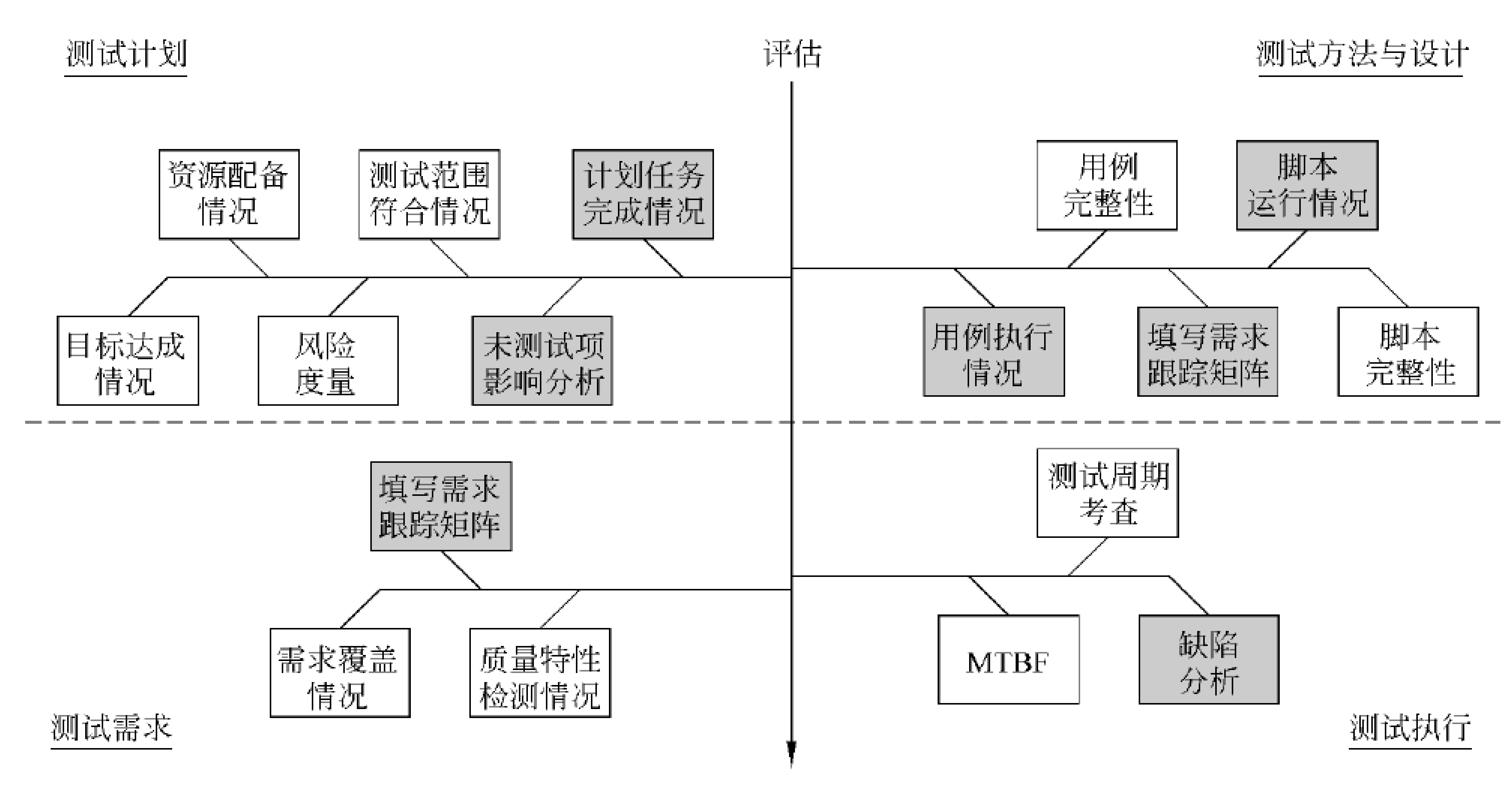


图 12-2　测试评估内容

更替要了解清楚，比如测试人员的抽离、测试时间的挤占等因素。评估这些变化是否对某些测试造成了影响，从而导致测试不足。

2．测试范围符合情况

评估测试计划中规定的测试项是否都已进行了有关测试，测试结果是否已经采集完成。有关测试范围的评估是一个难点，一般都是采取定性评估，对被测对象的边界进行粗线条的描述。一个有益的尝试是将测试对象分解，比如界面、函数、边界、路径等，统计被测对象有多少个界面、多少个函数、多少个边界和路径，这样以实现测试范围的量化，通过对量化指标的统计，我们可以得到这样一个直观的范围符合评估图(如图 12-3 所示)。

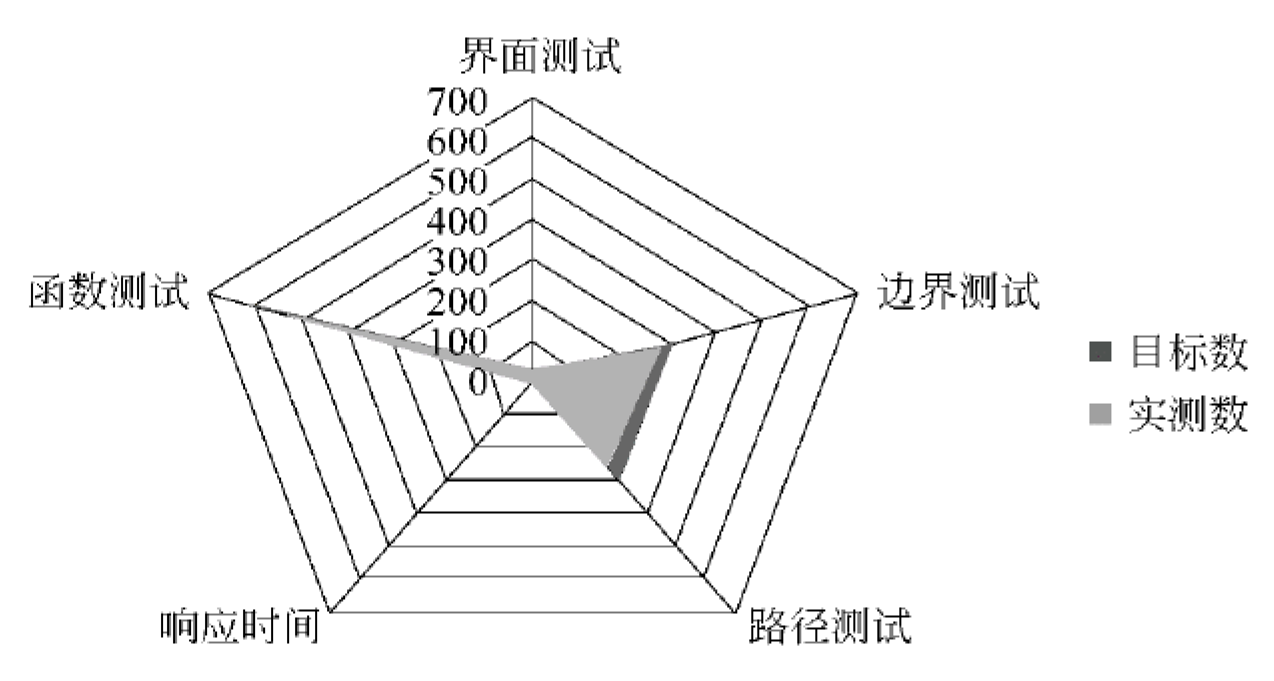


图 12-3　测试充分性雷达分析图

3．计划任务完成情况

对照测试计划的 WBS 工作分解，查验每一项工作的实施和完成情况，如有变更，则要追溯变更原因，并同时对变更后的实施和完成情况进行考查。

4. 目标达成情况

应对测试执行活动、测试报告、测试记录和测试问题报告进行评审，以确认测试执行活动的有效性，测试结果的正确性和合理性，是否达到了测试目标，测试文档是否符合规范。一般情况下，评审由软件测试方自行组织，评审细则也自行制定。其具体内容和要求应包括：

① 评审文档和记录内容完整性、正确性和规范性。

② 评审测试活动的独立性和有效性。

③ 评审测试环境是否符合测试要求。

④ 评审测试记录、测试数据以及测试报告内容与实际测试过程和结果的一致性。

⑤ 评审实际测试过程与测试计划和测试说明的一致性。

⑥ 评审未测试项和新增测试项的合理性。

⑦ 评审测试结果的真实性和正确性。

⑧ 评审对测试过程中出现异常的处理的正确性。

5. 风险度量

对预计的风险进行考查，弄清测试前所预计的风险是否在测试期间出现，造成了何种影响，对软件测试结果有什么影响。

6. 未测试项影响分析

对于未测试项也要进行严格的管理，要对未测试项对于系统相关部分的潜在影响做出全面评估，并体现到最终的测试报告中。

12.2.2 测试需求分析中的评估

测试需求分析的评估主要是以测试需求是否已覆盖产品需求、用户测试需求等为依据的考查。测试需求分析的过程会把原始的产品需求、与用户交流的测试需求、测试人员的补充需求转化为统一的测试需求，这些测试需求反映了被测系统的各种质量属性。在需求分析阶段评估，除了直接评估测试需求的转化情况外，还要查看所定义的质量属性是否得到充分验证。

在测试需求分析阶段，一个重要的输出产物是测试需求跟踪矩阵。我们在进行测试需求覆盖评估时，以需求跟踪矩阵为线索，上可以考查对产品需求、用户测试需求的覆盖，下可以考查测试用例对于测试需求的覆盖。

12.2.3 测试方法与设计中的评估

1. 用例(脚本)完整性

用例(脚本)的完整性主要评估用例(脚本)在测试期间有无发生变更，这种变更可以是修改、可以是删除、可以是新增，并评估这种变更对测试造成的影响。

2. 用例(脚本)执行情况

用例(脚本)的执行情况直接反映测试的进程和质量，在 11.2.2 节介绍测试执行过程

监控时曾经给出过有关测试用例的一些指标公式,但是此处与彼处不同,在评估阶段我们一般使用以下有关测试用例的考查指标:

$$\%\text{passed}(\text{测试用例通过率}) = T^p/\text{TCT}$$

$$\%\text{failed}(\text{测试用例失败率}) = T^f/\text{TCT}$$

其中,T^p 是指已执行的完全成功、没有缺陷的测试用例数,T^f 是已经执行且发现了被测系统缺陷的测试用例数,TCT 是测试用例的总数。

3. 填写需求跟踪矩阵

将测试用例数据植入需求跟踪矩阵,使其与测试需求建立对应关系,可以考查测试需求覆盖情况。一个有效的测试需求覆盖考查指标可以设置如下:

需求覆盖率 =(有派生用例的测试需求数)/(测试需求总数)

12.2.4 测试执行中的评估

1. 测试周期考查

查看测试用例执行情况,在最近一轮测试完成后,测试用例的通过率是多少,对于遗留缺陷开发团队的处理意见。

在测试周期考查中,我们还可以进行 ROI(投入产出比)评估,即统计最近一轮测试完成后,系统所发现的缺陷与前几轮测试发现的缺陷数做比较,如果出现明显地大幅下降,则意味着测试的 ROI 在降低。

2. MTBF(Mean Time Between Failed,平均故障间隔时间)

MTBF 是一个出色的度量指标,但是对于软件测试来说却比较难以使用,我们的经验做法是:统计最近的两次测试终止的间隔时间,或者以 M 个严重缺陷为单位统计,计算出现 M 的间隔时间,这个 M 的大小可随项目规模适度设置。

3. 缺陷分析

缺陷分析本质上是对缺陷中包含的信息项进行收集,汇总,分类之后使用统计方法(或者分析模型)得出分析结果。缺陷分析得出的结果可以用来度量软件开发过程中各阶段工作产品的质量,了解缺陷集中的区域,明晰缺陷发展趋向。对于软件过程的改进,软件产品的发布来说具有十分重要的参考价值。

1)缺陷趋势报告

“缺陷趋势报告”指出缺陷率并清楚地描述测试的状态。在测试工作早期,缺陷率快速上升,然后到达高峰,之后以较慢的速率下降。对于软件产品发布而言,发展趋势图是辅助决策的重要依据(如图 12-4 所示)。

从图 12-4 可以获得这样的信息,在 2009 年 1 月 3 日之后,HRMIS 的缺陷库中没有新增缺陷。这表示测试已经终止或者测试没有新增价值,测试可以考虑停止了。

如图 12-4 中反映的趋势显示,在项目开始时,发现和打开新缺陷的速率很快,但随着时间推移,该速率不断降低。打开的缺陷的趋势与新缺陷的趋势相似,但是稍微落后。因为打开的缺陷不断被修正和验证,关闭的缺陷的趋势随着打开的缺陷的修复和核实而不断增长。这些趋势证明工作是成功的。

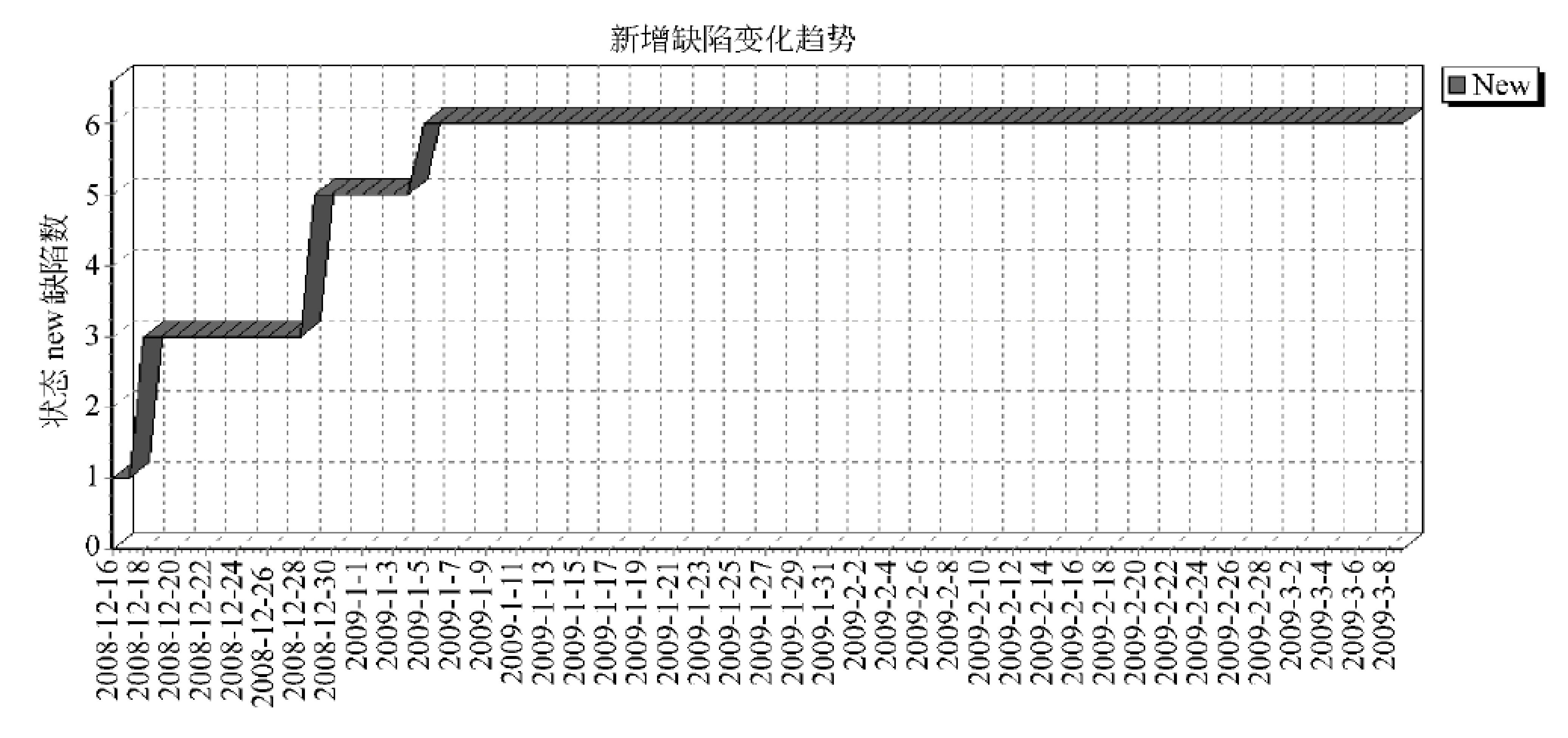


图 12-4　新增缺陷变化趋势(测试工具自动生成)

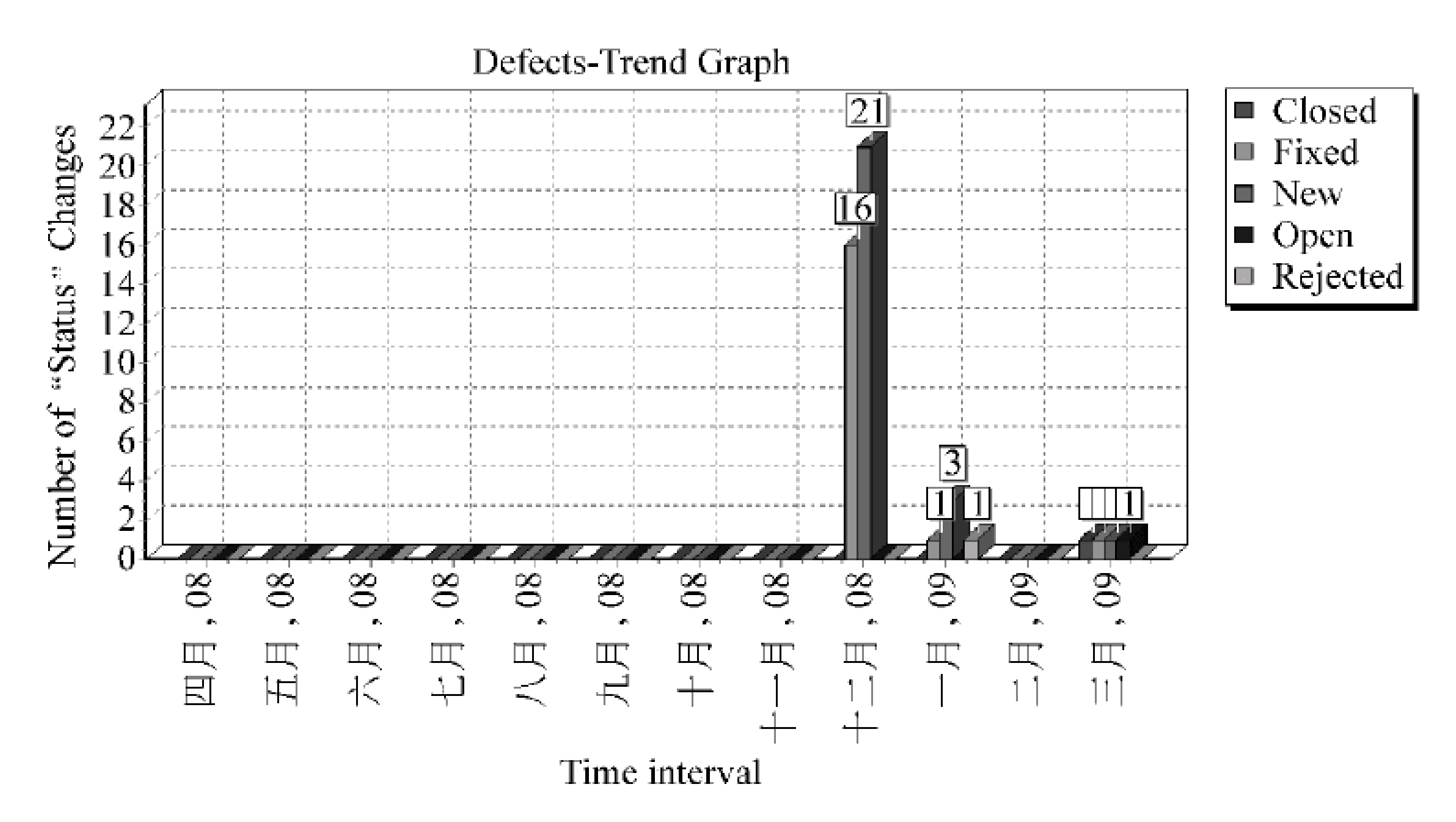


图 12-5　缺陷变化统计图(测试工具自动生成)

项目经理可通过持续观察缺陷趋势图,确保项目开发健康发展;测试负责人可通过持续观察缺陷趋势图,确保测试的有效性。同时,通过分析预测项目测试缺陷趋于零的时间,以制定产品质量验收和发布的时间。分析图表会告诉我们很多有价值的信息。比如说,可分析开发和测试在人力资源的配比上是否恰当,可以分析出某个严重的缺陷所造成的项目质量的波动。对于异常的波动,如本来应该越测试越收敛的,却到了某个点发现的故障数反而呈上升趋势,那么意味着往往有一些特殊事件的发生。通过对测试缺陷分析,能够给予我们很多改进研发和测试工作的信息。

2) 缺陷类别、状态统计

缺陷类别报告(如图 12-6 所示)可以向测试负责人提供以下信息,最近所发现的缺陷集中在哪一类别,是否有继续测试的必要,或者需要强化哪一部分的测试。如假设某项目统计发现最近一周的缺陷多集中在界面布局类,且状态多为可忽略状态,则测试负责人就

可以向项目经理汇报并建议结束软件的测试工作了。

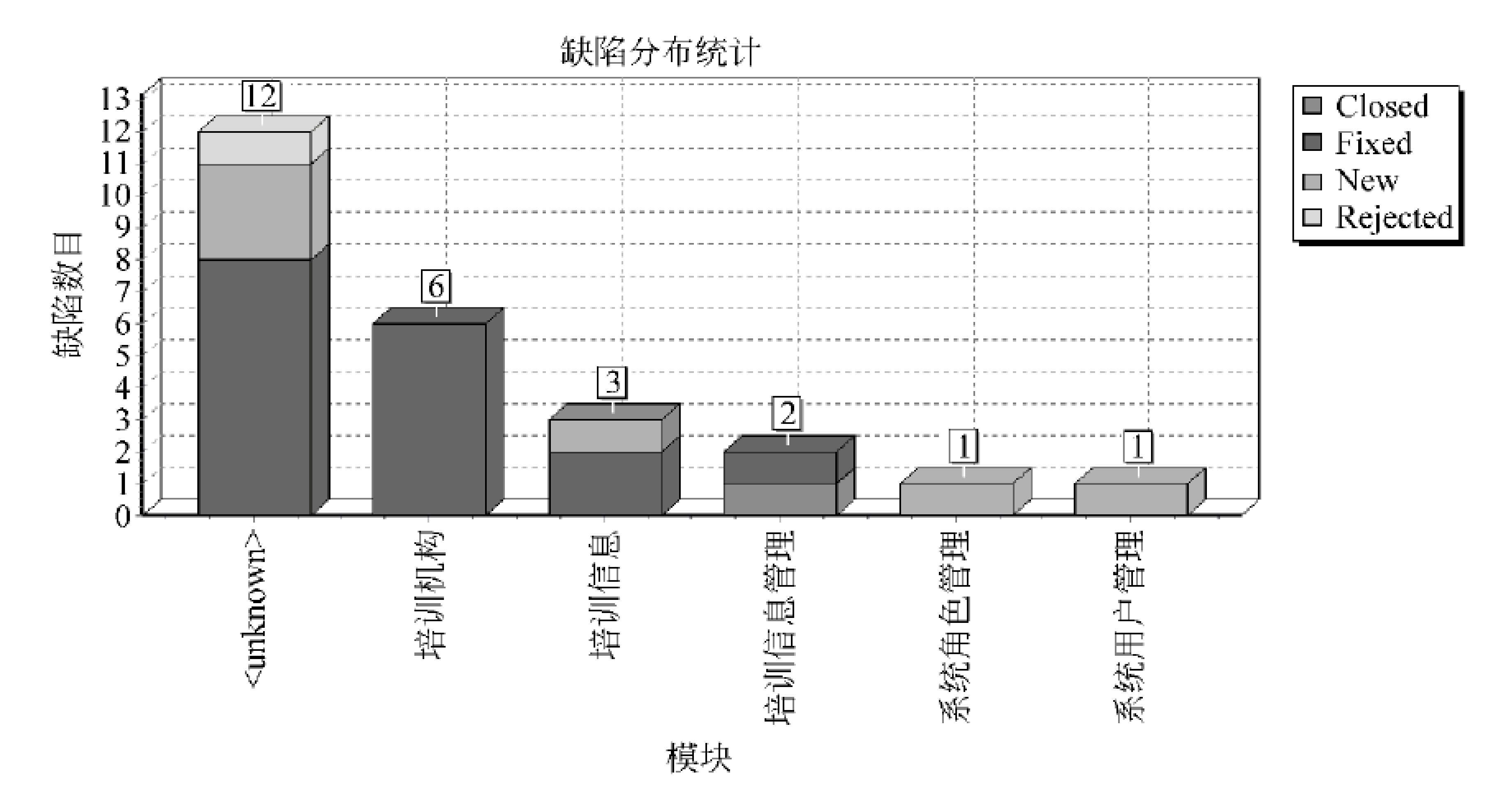


图 12-6 缺陷分布统计

3）缺陷状态、严重程度统计

缺陷状态分布报告(如图 12-7 所示)是在评估软件测试是否结束时一个重要的参考指标,一般企业都会在自己的测试停止标准内加这样一条:缺陷状态为中、高的都应关闭,经评审延迟处理的除外。因此,测试负责人在评估测试工作时,要及时查看缺陷的修改处理情况,保证所有的缺陷都得到有效处理。

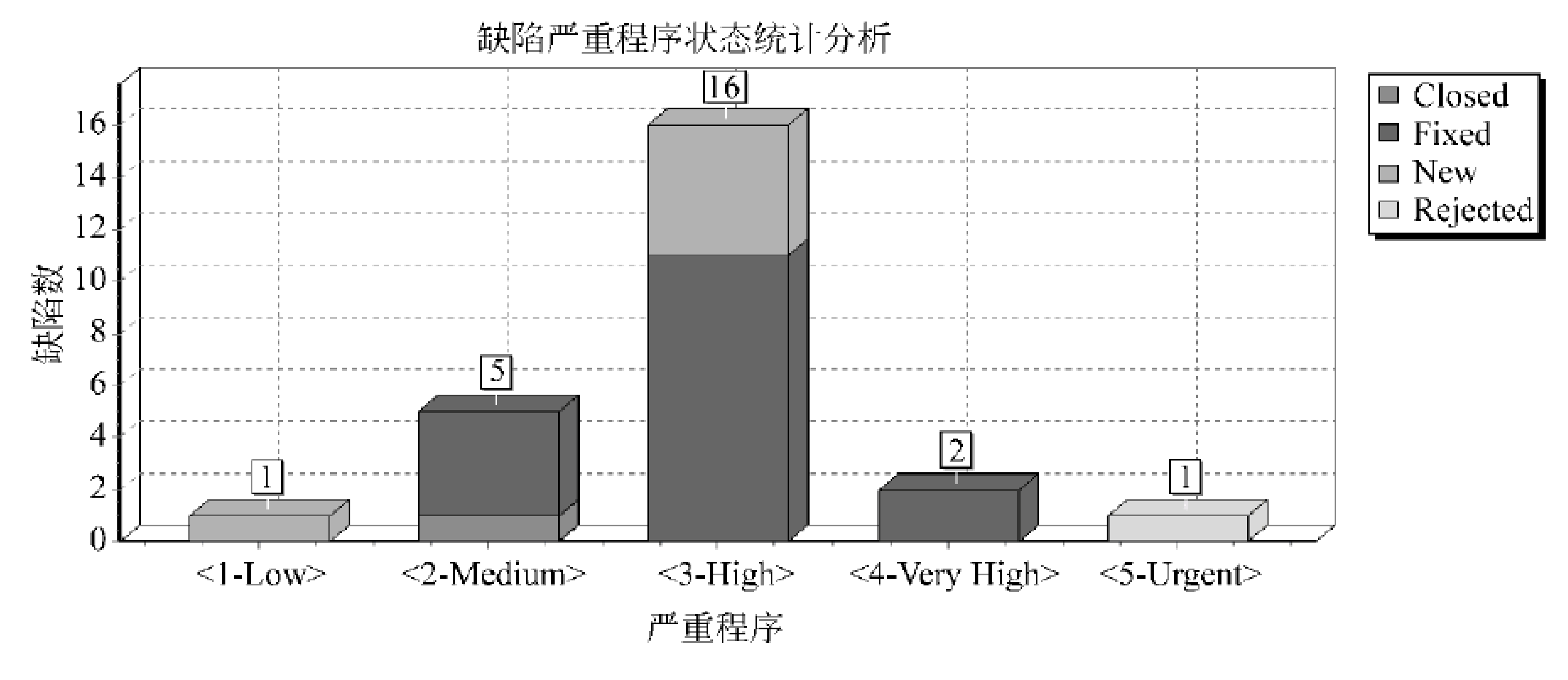


图 12-7 缺陷严重程度状态统计

有关测试执行的评估内容比较多,以上只是在系统层面的一个比较通用的评估方向,对于部分测试类型,比如单元测试,我们借助白盒测试工具还可以做出更加精确的测试覆盖率评估。

基于代码的测试覆盖率评估是对被测试的程序代码语句、路径或条件的覆盖率分析,评估测试中有多少代码已经执行和有多少代码有待执行。代码覆盖可以基于控制流(语句、分支或路径)或者数据流。在控制流覆盖中,目的是测试代码行、分支条件、代码路径

或软件控制流的其他元素。在数据流覆盖中，目的是测试数据状态在经过软件操作后仍然有效；例如，数据元素在被使用前已经定义过。

基于代码的测试覆盖率评估可以用下列公式计算。

$$测试覆盖率 = I^e / \mathrm{TIic}$$

其中，I^e 指执行的项（表示为代码语句、代码分支、代码路径、数据状态决定点或数据元素名）的数目，TIic（Total number of Items in the code）是代码中的项目总数。

【示例：应用 Rational TestRealTime 的 HRMIS 测试覆盖分析结果】

我们主要对 HRMIS 进行了语句、函数、判定三种方式的覆盖率测试分析（如图 12-8 所示），其中函数覆盖率 27.3%；语句覆盖率 21.5%；判定覆盖率 20.7%。通过运行测试用例可以发现测试中还有许多未覆盖到的函数和代码，需要进一步增加测试用例对程序进行更加充分的测试。

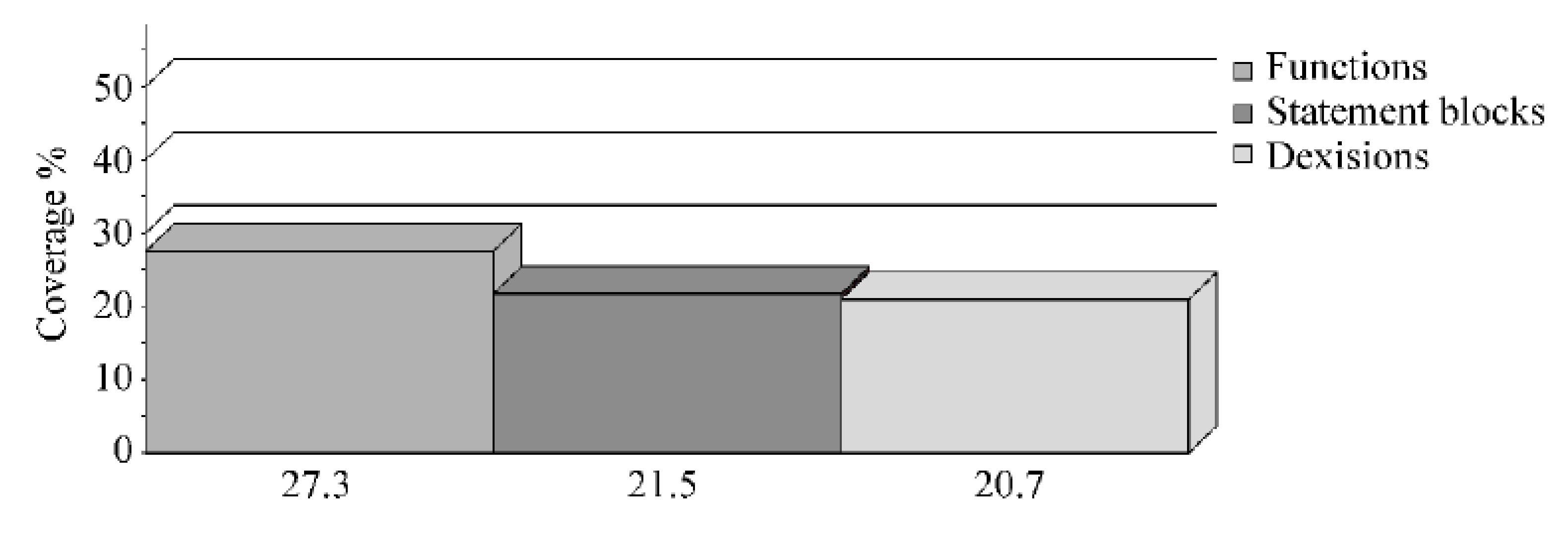


图 12-8　覆盖率分析

我们可以根据这个覆盖率分析结果进一步追踪到源程序中（如图 12-9 所示），在这个图中可以很直观地找到未覆盖到的程序和函数，根据这些未覆盖的信息可以很方便地确定需要增加哪些用例以提高覆盖率。

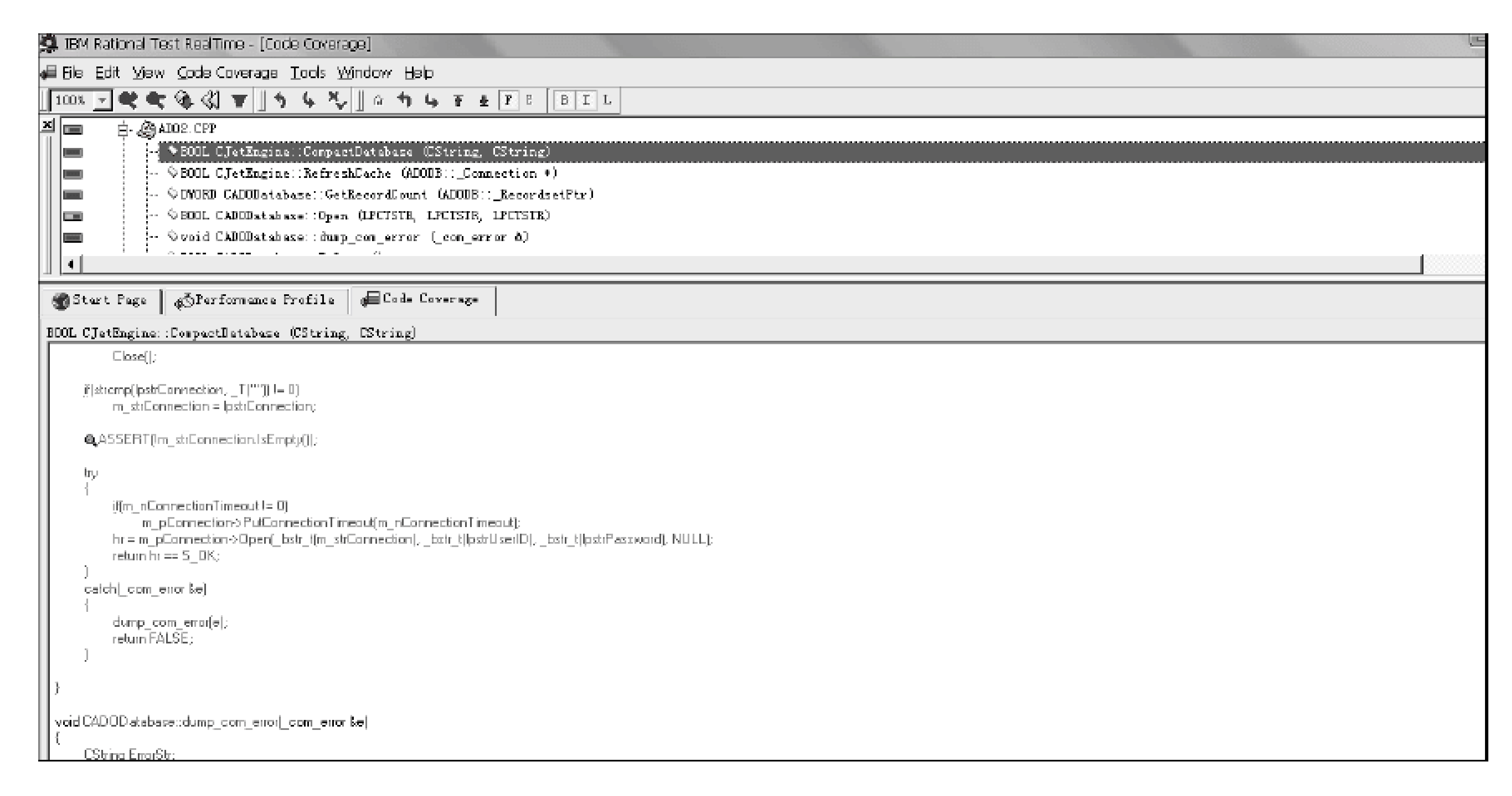


图 12-9　源码追踪

12.3 测试报告

对软件测试的全过程域评估之后，软件测试如果决定停止，则进入测试分析报告的编写阶段，也就是软件测试生命周期的最后一个阶段。本书在前面穿插介绍了单元测试、集成测试和系统测试的设计和执行，事实上对于前两者大多数时候是在开发团队内部进行的，它们的阶段性工作界限有时并不是十分清晰，而后者作为软件测试的最后一道工序，在软件测试过程中占据了优势地位，因而也获得了更多更广范的关注。因此，本节主要介绍系统测试报告的编写。

关于软件测试报告的编写也有很多不同的版本在业界流行，这也正说明不同的接收者对于软件测试报告存在不一样的信息诉求。本节以开发方内部的系统测试报告为例，全面介绍测试报告的内容要素和编写要求。这类测试报告旨在详细描述测试的过程和结果，并对发现的问题和缺陷进行分析，为纠正软件存在的质量问题提供依据，同时为软件验收和交付打下基础。因此，系统测试报告有一大部分的内容会来自软件测试评估的结果(如图 12-10 所示)。

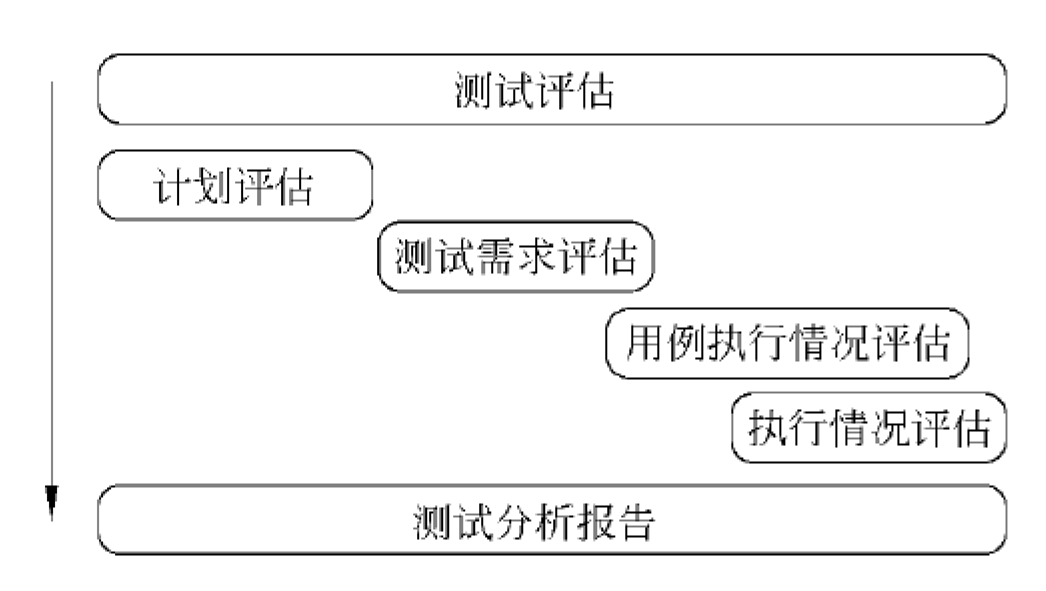


图 12-10 测试评估与测试报告

12.3.1 测试报告的一般性要求

一个好的测试报告，会使得所有阅读报告的人受益。因此，测试报告的编制要做到“信、雅、达”，具体来说就是文字流畅，无错别字，无俚语和俗语，表达准确，易理解，无歧义。报告中标点一般使用全角符号；对引用用户文档中原文或程序中菜单项、按钮说明等均使用引号标示；语句描述通顺完整。

测试报告是测试阶段最后的文档产出物，测试人员应该具备良好的文档编写能力，一份详细的测试报告包含足够的信息，包括产品质量和测试过程的评价，测试报告基于测试中的数据采集以及对最终的测试结果分析。

12.3.2 测试报告要素及实例

测试报告作为一份正式的文档，其编制也有一定的规范可供参考，下面我们结合 HRMIS 的测试报告的编写，来讨论一下这些要素。

1. 基本信息

1）版本控制信息

测试报告的首页应该有明确的版本控制信息，参考样例如下：

版本	作者	操作	时间	变更摘要

其中，版本类似软件版本编号，一般初稿定版为1.0，小版本变动，一般只增加小版本号，即1.1、1.2…，依此升级版本号，大的变更升级大版本号，即2.0、3.0…，依此类推。操作主要有新建、变更和审核等操作。

2）声明信息

软件测试的结果受测试环境等因素影响很大，因此一些必要的免责信息应该在测试报告中有所体现。另外，对于软件测试报告的使用及分发也应做出声明。

2. 引言

1）编写目的

本测试报告的具体编写目的，指出预期的读者范围。有关测试报告的编写目的有一些是基本固定的测试报告的作用，这部分可以直接套用，同时根据实际项目的需要，我们在编写测试报告时也以尊重事实的态度表达清楚。

前文中我们已经说过不同的接收者对于软件测试报告存在不一样的信息诉求，所以在测试报告中增加一部分类似于阅读指南性的内容是有必要的，这样可以更好地激发有关人员对阅读测试报告的兴趣，同时也节约时间。通常，用户会对测试结论部分感兴趣，开发人员对缺陷结果以及分析更加关注，而项目管理者和测试负责人对测试成本、资源和时间比较关心，而高层经理希望能够阅读到简单的图表以更加直观地了解项目信息，并且能够与其他项目进行横向比较。因此，这一部分可以具体描述什么类型的人可参考本报告×××页×××章节，这样导向型的报告信息，可以使接收者更快地取得其关心的信息。

2）项目背景和系统简介

项目背景介绍项目的建设目标，如果是独立第三方测试也可以包含部分委托信息。一般这部分内容可以直接从需求或者招标文件中引用即可。

系统简介一般是从系统设计说明书引用，注意多引用一些必要的框架图和系统拓扑图，因为一图胜千言，图片包含的信息有时是文字所不容易表达清楚的。另外，图文并茂的文档更容易引起阅读者的兴趣。

3）术语和缩写词

列出设计本系统/项目的专用术语和缩写语约定，对于技术相关的名词与多义词一定要注明清楚，以免阅读时产生歧义。

4）参考资料

对于软件测试来说，可作为参考资料的可以有项目文档和测试特有的相关质量标准

或者行业规范等。

① 一般可供参考的资料有需求说明书、设计说明书、测试用例、操作手册以及其他项目文档等。

② 测试使用的国家标准、行业指标、公司规范和质量手册等。

【HRMIS 系统测试报告——引言部分实例】

1　引言

1.1　编写目的

编写本测试报告的目的是:

- 通过对测试结果的分析,得到对软件质量的评价。
- 分析测试的过程、产品、资源、信息、为以后制定测试计划提供参考。
- 评估测试执行和测试计划是否符合。
- 分析系统存在的缺陷,为修复和预防缺陷提供建议。

本报告的预期读者:高级管理者、项目(产品)经理、开发部门负责人、测试中心、开发人员。

1.2　项目背景

- 项目名称:人力资源管理系统(HRMIS);
- 最终用户:××××;
- 监理单位:××××;
- 相关供应商:××××;
- 软件关联关系:略,系统外部接口测试不在本次测试范围内。

1.3　客户信息

客户信息	
客户名称	—
客户地址	—
被测软件名称	人力资源管理系统(HRMIS)
被测软件版本号	0.2
被测软件标识	无
被测软件描述和状态	

1.4　测试机构

测试机构信息	
测试机构	××××
测试机构地址	
测试报告批准人	

续表

测试负责人	—
测试人员	—
被测软件提交时间	20××年 12 月 28 日
被测软件接受时间	20××年 12 月 28 日
测试时间	20××年 12 月 28 日至 20××年 1 月 19 日
测试地点	—
测试方法	黑盒测试、自动化测试相结合

1.5　术语和缩写词

略。

1.6　参考资料

在软件评测活动中确认移交的包括业务需求及用户操作说明书，文件列表如下：

文档编号	文档名称	版　本
HR-SRS-0200-200811	人力资源管理系统需求说明书	2.0
HR-SDS-0100-20081229	人力资源管理系统设计说明书	1.0
P2008-12-001-RPT01-01	人力资源管理系统测试计划	1.0

3．测试概要

测试概要顾名思义，主要是对整个测试做一个简明扼要的说明，包括测试的目的、测试范围、测试环境等信息。

1）测试目的

简要描述测试的目的，例如是要测试系统的功能，还是系统的性能。

2）测试范围

列出测试类型及所有要测试的功能模块。

3）测试用例设计

简要说明测试用例所使用的用例设计方法。例如：等价类划分、边界值、因果图等(有关方法的介绍请参阅第 10 章有关内容)。

4）测试环境与配置

详细描述测试所使用的测试环境及其软件、硬件配置。一般对配置过程没有详细描述，但是对于配置复杂的测试环境建议给出配置过程，或者以附件的形式另外说明。

5）测试方法和工具

说明测试中采用的方法和工具。系统测试主要是以黑盒测试为主，测试方法可以描述测试的重点和采用的测试模式，这样可以一目了然的知道是否遗漏了重要的测试点和关键块。

测试工具为可选项，有则有之，无则无之。注意要注明是自研还是商业工具，所用软件的版本信息等，在测试报告发布后要避免工具的版权问题。

6）测试组织

介绍测试组的构成和领导关系，一般以框图的形式出现。

【HRMIS 系统测试报告——测试概要部分实例】

2　测试概要

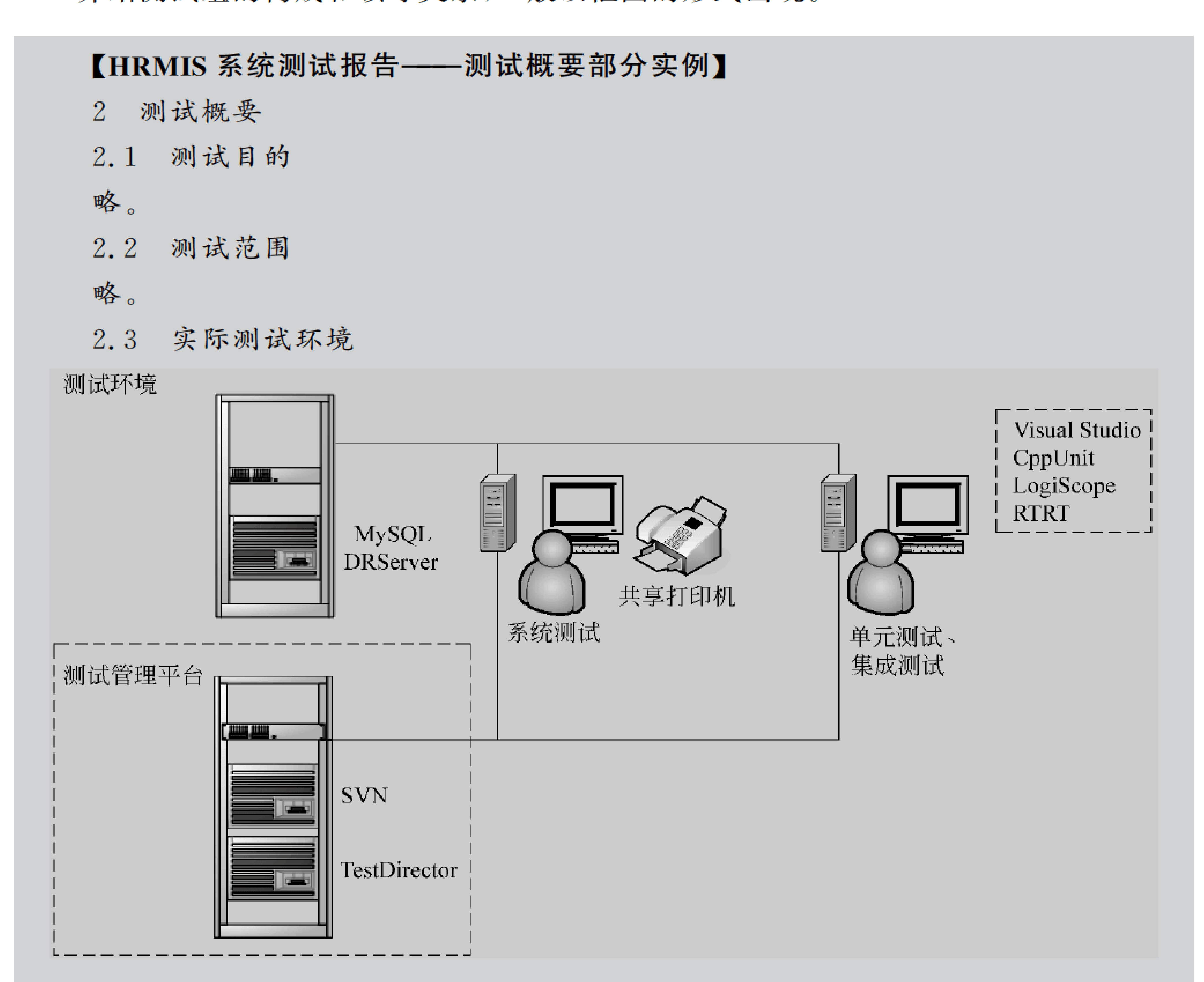
2.1　测试目的

略。

2.2　测试范围

略。

2.3　实际测试环境



测试环境仿真模拟客户的生产环境，构建 Client/Server 服务模式。为了对测试过程进行有效地管理和控制，测试期间使用 SVN 配置管理工具对部分测试产出物进行版本控制，使用 TD8.0 对测试过程进行管理，如测试需求管理、用例开发及管理和缺陷跟踪等。构建上述测试环境的部分软件、硬件清单如下：

2.3.1　硬件环境

数据库服务器硬件环境	
CPU 种类及数量	Intel Xeon 主频 2.0GHz，2 个
内存	2GB
硬盘	160GB
网卡	RTL8139 10/100 Mbps Ethernet PCI Adapter

客户端硬件环境	
CPU 种类及数量	Intel Pentium 4,主频 1.1GHz,1 个
内存	512MB
硬盘	80GB
网卡	100MB

2.3.2 软件环境

数据库服务器	
操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
数据库管理系统	MySQL 6.0

客户端 1 软件环境	
操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
MySQL 数据库访问接口	MyODBC 3.5.1
开发工具	Visual Studio 6.0
白盒测试工具	CppUnit1.12.0
	Logiscope 6.1
	Rational TestRealtime

客户端 2 软件环境	
操作系统及补丁	Windows XP SP2
防病毒软件	卡巴斯基 6.0
MySQL 数据库访问接口	MyODBC 3.5.1

测软件系统	
软件名称	人力资源管理系统(HRMIS)
软件版本	0.2
说明	需要在 2 台客户机安装

2.4　测试依据

GB/T 16260.1—2006《软件工程　产品质量》

GB/T 17544—1998《信息技术 软件包 质量要求和测试》

GB/T 18905.5—2002《软件工程 产品评价第 5 部分 评价者用的过程》

参考资料中列示文件

2.5　软件说明

人力资源管理系统 HRMIS 是一个面向中小型软件企业人力资源管理的小型应用,以计算机信息处理技术实现企业或组织的员工信息管理,如基本资料管理、考勤与考评管理、薪酬管理、合同管理以及培训管理等。

2.6　测试过程及方法

本次测试我们遵循一般软件测试过程展开软件测试活动,确立了测试需求、测试计划、测试用例开发、测试执行、回归测试的软件测试过程。各个测试阶段的里程碑列示如下:

(1) 测试需求里程碑:形成《需求跟踪矩阵》

(2) 测试计划里程碑:批准《人力资源管理系统测试计划》。

(3) 测试用例设计里程碑:测试用例库组建完成,《测试用例报告》获得批准。

(4) 实施测试里程碑:测试用例执行完毕,缺陷库组建完成,《缺陷报告》获得批准。

(5) 评估测试里程碑:系统测试报告批准。

本次测试活动经客户、开发方和测试方确认的测试范围为文档测试、功能测试、性能测试,其中功能测试采用黑盒测试法,手工执行测试用例来对系统各个功能模块进行检测,性能测试应用性能测试工具创建自动化测试脚本进行自动化测试。

本次测试所使用的测试工具分为测试管理工具和性能测试工具,其中测试管理工具为 TD8.0,性能测试工具为 LoadRunner 9.1。

2.7　测试组织

略。

4. 测试结果

1) 测试执行情况与记录

描述测试资源消耗情况,记录实际数据。

2) 测试时间

列出测试的跨度和工作量,最好区分测试文档和活动的时间。该数据可供过程度量使用,为过程改进提供依据。

对于大系统/项目的测试来说最终要统计测试资源的总投入,以便管理者清楚地知道究竟花费了多少人力去完成测试。在数据汇总时可以统计个人的平均投入时间和总体时间、团队平均投入时间和总体时间,还可以计算每一个功能点所花费的时/人。另外,统计计算文档生产率和测试执行率也是该部分中的有关过程度量的重要内容。

3）测试版本

给出测试的版本，如果是最终报告，可能要报告回归测试次数。列出表格清单则便于知道某个子系统/子模块的测试频度，对于多次回归的子系统/子模块应引起开发者关注。

4）测试覆盖分析

（1）需求覆盖率。这里所说的需求覆盖率是指经过测试的需求/功能和需求规格说明书中所有需求/功能的比值，通常情况下要达到100%的目标。

需求编号(ReqID)	测试类型	是否通过	备注
		P、N、N/A	

根据测试结果，按编号给出每一测试需求的通过与否结论，一般有三种状态，N表示不通过，P表示部分通过，N/A表示不可测试或者用例不适用。事实上，从第二章也已经了解到需求跟踪矩阵列出需求与用例的一一对应情况，以避免遗漏，此矩阵表的作用为传达需求的测试信息以供检查和审核。

$$需求覆盖率 = (N+P)/R \times 100\%$$

其中R表示需求总数。

（2）测试覆盖。

功能编号(FunID)	用例数	执行	未执行	未测漏测原因分析	备注

上表是功能-用例关系矩阵，通过这个矩阵可以方便地对待测功能进行检查，以此计算的测试覆盖率可以按照如下公式进行：

$$测试覆盖率=用例执行数/用例总数\times 100\%$$

（3）缺陷的统计与分析。有关缺陷统计分析的内容前面介绍的比较多，在此不再赘述。

【HRMIS系统测试报告——测试结果部分实例】

3　测试结果

3.1　测试执行情况说明

系统测试期间确认的测试任务主要有功能测试和性能测试，其中功能测试由功能测试组于2008年12月28日至2008年12月31日进行，主要测试的功能有员工信息管理（含工作履历、合同、培训记录、部门信息调动、变更记录）和培训信息管理。2009年1月1日至2009年1月2日为缺陷确认和修改，2009年1月3日至2009年1月8日进行首轮回归测试。

2009年1月1日对系统控制部分进行测试，主要是系统用户管理、组织机构管理、权限控制。

至本报告编写之日，以上功能测试的测试用例已全部执行完毕，并且达到95%的通过率。

2009 年 1 月 3 日至 2009 年 1 月 6 日由性能测试组对系统性能进行了测试，并按照计划于 2009 年 1 月 7 日至 2009 年 1 月 8 日对系统进行了优化调整，2009 年 1 月 13 日再次对系统做了性能测试。

至本报告编写之日，所有性能测试用例均执行完毕，性能测试期间没有失败事务和错误出现，全部性能指标均达标。

3.2 功能测试部分

本节所述通过、基本通过、不通过特指以下含义：

- 通过：完全符合要求。
- 基本通过：符合部分要求或部分符合要求，且不影响该项目的总体要求；存在缺陷，但是经客户确认可以暂缓修改；存在有待于相关业务处室确认的问题。
- 不通过：不符合要求；或者部分不符合要求，而且影响该项目的总体功能要求。

3.2.1 主要业务功能

一级模块	二级模块	三级模块	覆盖/结果	遗留缺陷
员工信息管理	新增员工信息		基本通过	4
	修改员工信息		通过	
	删除员工信息		通过	
	部门调动		通过	
	合同管理	合同签订	通过	
		合同修改	通过	
		合同删除	通过	
	工作履历	添加工作履历	通过	
		修改工作履历	通过	
		删除工作履历	通过	
	参加培训记录		通过	
	信息变更历史记录		通过	
培训信息管理	培训信息添加、修改和删除		不通过	1
	培训机构添加、修改和删除		通过	
	培训信息的培训机构变更		通过	
系统管理	系统用户添加、修改和删除		通过	
	系统用户权限设置		通过	

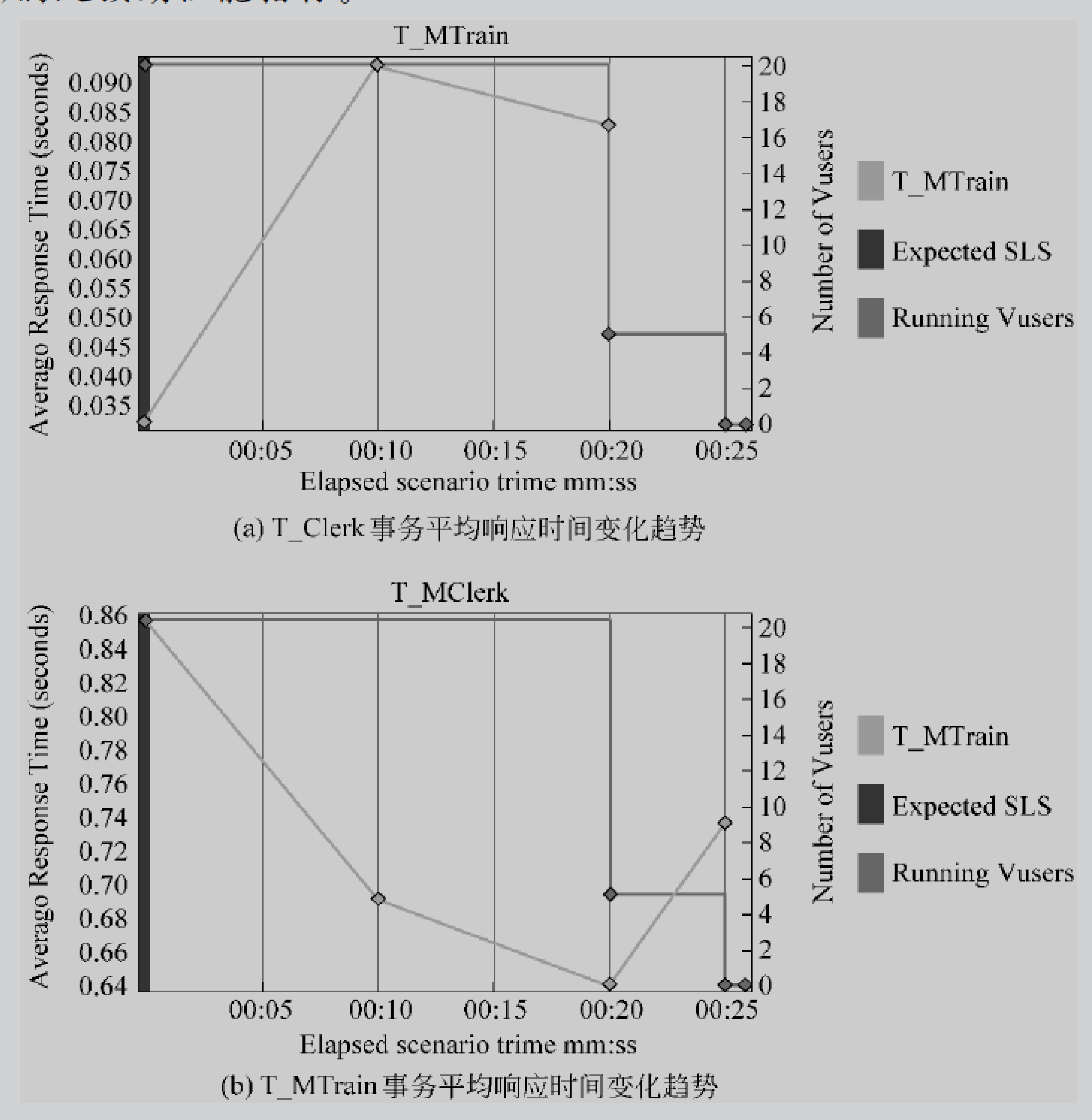
3.2.2 其他测试项目

无。

3.3　性能测试部分

（1）平均响应时间

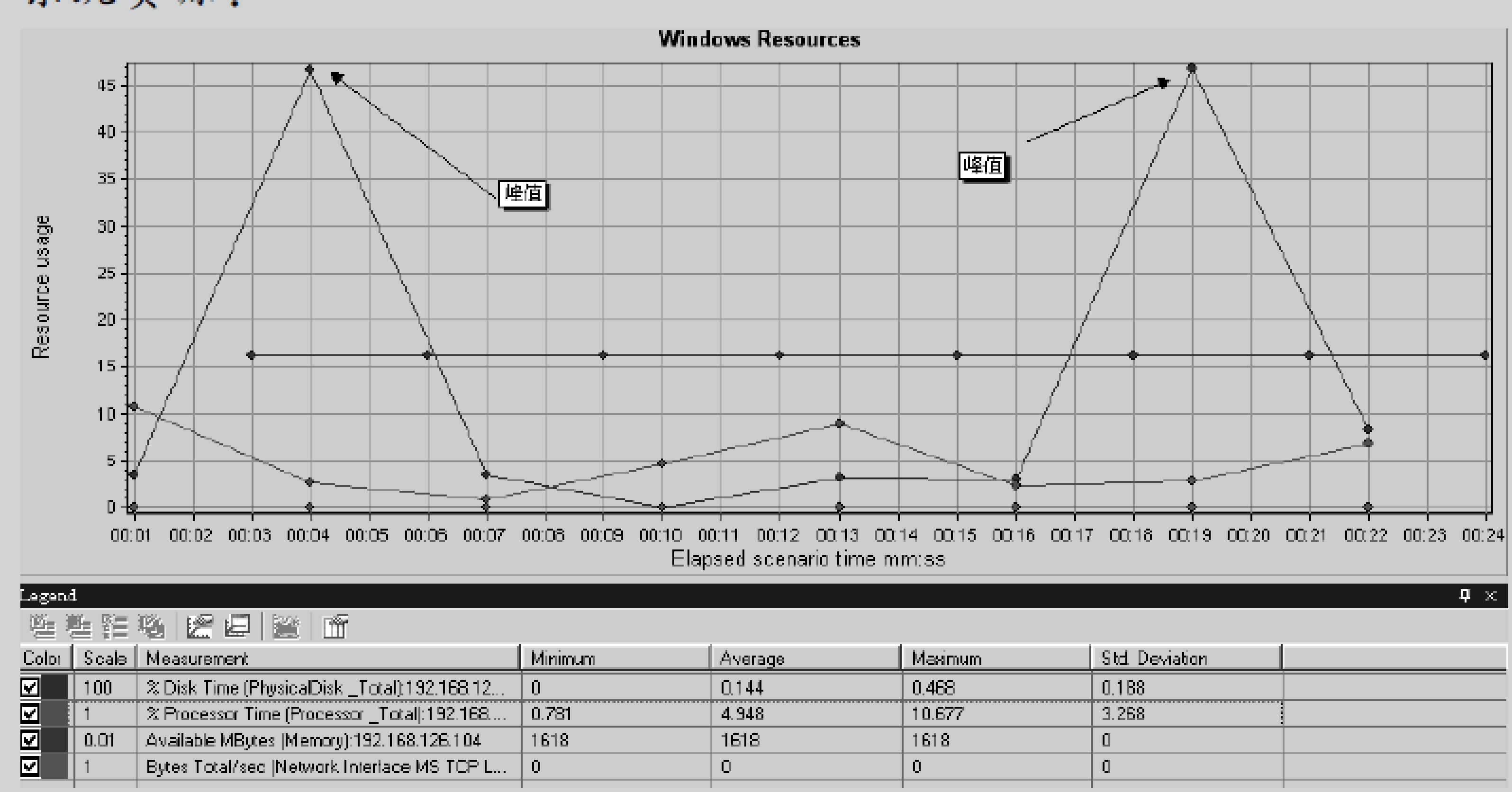
并发用户为 20 时，修改员工信息、修改培训机构信息这几个功能点的平均响应时间低于 15s，满足预期性能指标。



(a) T_Clerk 事务平均响应时间变化趋势

(b) T_MTrain 事务平均响应时间变化趋势

（2）系统资源：



Color	Scale	Measurement	Minimum	Average	Maximum	Std. Deviation
☑	100	% Disk Time (PhysicaDisk _Total):192.168.12...	0	0.144	0.468	0.188
☑	1	% Processor Time (Processor _Total):192.168....	0.781	4.948	10.677	3.268
☑	0.01	Available MBytes (Memory):192.168.126.104	1618	1618	1618	0
☑	1	Bytes Total/sec (Network Interface MS TCP L...	0	0	0	0

CPU：从曲线图上可以看出（见附件 1），数据库服务器在整个测试期间消耗变化不大，比较平稳。

内存：可用内存基本没有变化，没有因内存不足产生的内存页交换。

(3) 稳定性

测试过程中,没有产生宕机现象;

各事务响应时间基本上不随着时间的变化而加长;

20 并发用户时,无失败事务。

优化建议：无。

3.4　遗留缺陷影响

略。

5. 测试结论

测试结论无疑是整份测试报告的焦点,本质上测试结论就是对上述过程、缺陷分析之后的总体评价和决断。

1）测试结论

测试结论可以从以下方面进行阐述,例如测试执行是否充分(可以增加对安全性、可靠性、可维护性和功能性描述);测试风险的度量(风险有无出现,控制措施是否有效);测试目标是否达成等。

2）建议

对系统存在问题的说明,描述测试所揭露的软件缺陷和由此带来的限制;对缺陷修改和产品优化改进的建议;对软件过程改进方面的建议。

【HRMIS 系统测试报告——测试结论部分实例】

4　测试结论

××××单位受××××公司的委托于 20××年 12 月 28 日至 20××年 2 月 7 日根据 GB/T 16260(2006)《软件工程 产品质量》、GB/T 17544(1998)《信息技术 软件包 质量要求和测试》、GB/T 18905.5《软件工程 产品评价第 5 部分 评价者用的过程》对×××××公司开发的“人力资源管理系统 HRMIS”在功能、性能(可靠性、效率、资源占用率)和用户文档等三个方面进行了系统测试。

测试结果表明“人力资源管理系统 HRMIS”大部分功能达到了所提供资料中(需求说明书、系统设计说明书)关于产品功能的说明,功能相对完整。但是,在测试过程中也发现了现有软件系统与业务需求不一致的地方,比如培训信息录入后不能修改、员工记录查询不可用,同时测试期间也发现了系统中存在高、中、低各个层级的缺陷,在委托方、测试方、开发方的共同努力协调下,与各个业务处室相关业务人员进行了确认,对所有已发现的缺陷进行了评估及确认,并对确认的问题进行集中修改,经过多次回归测试验证证明绝大多数已确认的问题已经获得修正,使软件质量有了一个新的提高。

5　建议

HRMIS 的操作菜单需要双击才能打开对应的功能界面,建议改为单击,使操作更简单。

附件 A　测试结果统计表

被测软件名称	人力资源管理系统（HRMIS）					版本号		V0.2	
业务功能覆盖	100%	缺陷总数	36	功能总数	23	缺陷率	1.57		
缺陷分析	按类型划分						按严重程度划分		
	缺陷分类	功能缺陷	界面缺陷	配置缺陷	其他缺陷	建议	高	中	低
	缺陷数	20	8	2	6	1	18	9	9
	总占比	56%	22%	6%	17%		50%	25%	25%

说明：

（1）业务功能覆盖率：以3.2功能测试结果列出的功能为计算依据。

（2）缺陷总数：以3.2功能测试结果列出的功能为计算依据(扣除建议性缺陷；扣除文档缺陷；扣除建议)。

（3）缺陷率：缺陷总数(36)/功能点(23)。

以上所述内容多数是我们站在第一方测试这样一个角度对测试报告编写的要素说明，对于第三方软件测评机构，测试报告编制还有更加严格的一些要求，例如：

① 测试报告需要详细说明测试机构、委托测评方的信息。

② 检测报告的唯一性标识(如系列号)和每一页上的标识，以确保能够识别该页是属于检测报告的一部分，以及表明检测报告结束的清晰标识。

③ 检测报告或校准证书批准人的姓名、职务、签字或等效的标识。

④ 相关时，结果仅与被检测物品有关的声明。

12.3.3 测试报告的管理

通常，测试报告供内部测试完毕后使用，且因测试报告本身包含一些涉及商业机密的信息，一旦泄露可能会对企业或者委托方造成重大损失，所以测试报告的发布一般都有严格的流程加以控制。

1. 保密性

测试报告实行保密化管理，限于企业内部人员使用的设置中级保密，如果可供用户和更多的人阅读，密级降为低，高密级的测试报告适合内部研发项目以及涉及保密行业和技术版权的项目。

2. 分发控制

很多第三方软件评测机构都对测试报告的分发做出了严格规定，例如要求授权签字人签名、一式两份(一份备案，一份分发)等。

3. 使用要求

测试报告的使用者必须有相应的授权，并且测试报告在引用时必须全文引用方才有效，严禁断章取义。

本章小结

软件评估是软件测试中一个相对薄弱的环节，尤其是在实际应用领域。本章试图从最佳实践的角度，不苛求权威地介绍了软件测试生命周期的最后一个阶段软件评估。软件评估实际上包含两部分内容，一是评估，二是报告。充分的评估是写好测试报告的基础。

软件评估是整个测试过程的全域考查，是站在过程决策角度的一种度量，具体可以细分为计划的评估、需求的评估、用例和执行的评估等。软件测试报告部分，从第一方测试的角度全面剖析了主流测试分析报告的编制要素，并给出了HRMIS的测试报告实例。

PART 4

第四篇 测试管理与过程改进

本篇讨论测试过程的组织与管理，主要分为两部分内容—测试组织和管理、测试过程改进。

测试组织与管理部分分别介绍了测试过程组织和测试管理包括的内容，以及如何进行测试过程组织和管理；

作为测试组织和管理的一部分内容，测试过程改进是为了更好地进行测试组织和管理。本篇的测试过程改进部分主要介绍了测试度量和测试过程改进两部分内容。测试度量部分主要介绍了测试度量的概念、度量内容、度量分类、度量过程；测试过程改进部分主要介绍了测试过程改进内容、改进过程、改进策略、改进注意事项，以及目前常用的测试过程改进模型。

本篇内容有助于读者理解软件测试的组织与管理，并结合自己的实际情况，制定出适合自己的测试过程组织形式和管理方法，提高软件测试效率，保证软件测试的质量。

本篇名词解释

软件过程改进(software process improvement)：实施对象就是软件企业的软件过程，也就是软件产品的生产过程，也包括软件维护之类的维护过程，而对于其他的过程并不关注。

度量构造(measurement structure)：是指将满足指定的信息需求的可度量事物联系起来的详细结构，度量构造描述了如何将软件的相关属性量化，然后转化成提供决策者使用和分析的指示器，最终得出信息产品。度量构造可以包括基本度量、派生度量和指示器三种度量类型或层次。

IDEAL：是一个组织用于启动、规划和实现过程改善措施蓝图的模型，它概括了建立一个成功的过程改善项目的必要步骤，其中 I 代表 Initiating（启动阶段），D 代表 Diagnosing（诊断阶段），E 代表 Establishing（建立阶段），A 代表 Acting（实施阶段），L 代表 learning（学习阶段）。

检查单（CheckList）：软件质量管理活动中最常用的工具之一，主要是列出被检查对象需要检查的各项内容，检查单的作用是提醒检查人员检查哪些内容，避免遗漏。

PDCA：是由美国统计学家戴明博士提出来的，它反映了质量管理活动的规律，其中 P（plan）表示计划；D（do）表示执行；C（check）表示检查；A（action）表示处理。PDCA 循环是提高产品质量，改善企业经营管理的重要法，是质量保证体系运转的基本方式。

TMM：测试成熟度模型（test maturity model）模型，提出测试的 5 个不同级别，关注于测试的成熟度模型，每个等级代表着一个成熟的测试过程。TMM 描述了测试过程，是项目测试部分得到良好计划和控制的基础。

TPI：测试过程改进模型（test progress improvement），通过 TPI 模型，能够确定测试过程的当前状态、下一个需要进行过程改进的地方，以及推荐的改进步骤。

CHAPTER

第 13 章 软件测试过程组织与管理

随着软件开发规模的增大、复杂程度的增加,以寻找软件中的错误为目的的测试工作就显得更加困难。然而,为了尽可能多地找出程序中的错误,生产出高质量的软件产品,加强对测试工作的组织和管理就显得尤为重要。

软件测试的组织与管理的目的是要对软件产品的整个测试流程进行控制和管理,提高软件测试机构的软件测试能力和测试工作效率,确保软件测试的质量。

测试的组织主要包括测试人员组织和测试过程组织。其中测试人员组织指的是组织结构、人员组成、组织规模等;测试过程组织主要包括测试过程规划、测试实施、过程改进等。软件测试管理主要包括过程管理、配置管理、风险管理等。其中,过程管理主要对测试过程中的测试活动和测试资源进行管理;配置管理主要对测试过程中产生的各种工作产品进行管理,测试工作产品包括测试计划、测试说明书、测试用例、测试报告、缺陷报告等,对测试工作产品的管理主要是测量和分析被测软件产品,检查和评审软件测试工作产品,收集质量分析和对产品进行决策所需要的数据;风险管理主要是对测试过程中的风险进行分析,并制定针对性地措施,防止风险发生,或者把风险降到最小。

13.1 软件测试组织

13.1.1 人员与团队

测试人员组织主要在组织结构、人员组成、组织规模等几个方面进行考虑。

1. 组织结构

管理大师彼得·德鲁克说:"组织管理的目的就是为了使人们能为实现目标而有效地工作,为此必须设计和维持一种职务结构。"因此,要做好软件测试工作,在软件公司内部必须建立一个专门负责软件产品测试的组织,并配备负责软件公司全面测试工作的测试管理人员和配备一定数量的具有测试理论、掌握软件测试技术的专业测试人员。

可以根据软件公司规模的大小设置软件测试组织的组织架构、隶属关系和确定软件测试人员的数量。常见的软件测试组织结构如下：

（1）测试部门是软件公司的一个独立部门，与软件开发部门同一个级别，专门负责公司所有软件产品的测试工作，配备一定数量专门从事软件测试工作的测试人员。

（2）测试机构是隶属于开发部门的一个测试组。有的软件公司没有单独设置软件产品测试机构，而是在开发部下成立了一个测试组，专门负责公司软件产品的测试。这种模式适合于一些规模比较小的软件公司，由于受到自身人力、物力等方面的限制，只能建立一个规模小的测试队伍。

2. 人员组成

一个好的测试团队首先要有好的带头人，他必须具有极为丰富的开发经验，对开发过程中常见的缺陷或错误了然于胸。此外，他还应具有亲和力和人格魅力。其次，测试团队还应有具备一技之长的成员，如对某些自动化测试工具运用娴熟或能轻而易举地编写自动化测试脚本。另外，测试团队还应有兼职成员。如验证测试实施过程中，同行评审是最常使用的一种形式，这些同行专家就属于兼职测试团队成员的范畴。测试团队里往往不乏缺少软件开发经验的新手，可以安排这部分人去从事交付验证或黑盒测试之类的工作。

表 13-1 中的内容为常见的测试人员组成及对应的职责，可以根据自己的情况来确定测试团队。

表 13-1　测试团队角色

工作角色	具　体　职　责
测试项目负责人	管理监督测试项目，提供技术指导，获取适当的资源，制定基线，技术协调，负责项目的安全保密和质量管理
测试分析员	确定测试计划、测试内容、测试方法、测试数据生成方法、测试（软、硬件）环境、测试工具，评价测试工作的有效性
测试设计员	设计测试用例，确定测试用例的优先级，建立测试环境
测试程序员	编写测试辅助软件
测试员	执行测试、记录测试结果
测试系统管理员	对测试环境和资产进行管理和维护
配置管理员	设置、管理和维护测试配置管理数据库

3. 组织规模

测试组织规模需要根据自己的实际情况来确定，受多个因素的影响，如企业文化或测试成熟度、测试需求范围、工程师技能水平、测试工具及应用水平、业务知识水平、组织形式、测试工作介入时间等。

对于组织规模的确定，可以采用以下几种方法，其中具体数据（如比例、计算因子等）的确定可以参考组织内的历史数据分析获得，表中所列的数据仅作为示例性数据供大家参考，不具有实际意义。

① 开发比例法。根据开发人员数量按照一定比例来确定测试工程师的数量。开发人员指进行设计、开发、编译以及进行单元测试的人员。表 13-2 是一个开发比例法的示例。

表 13-2　开发比例法示例

开发类型	开发人员	比例	测试组规模
商业产品(大型市场)	30 人	3∶2	20
商业产品(小型市场)	30 人	3∶1	10
单个客户端的应用开发	30 人	6∶1	5
单个客户端开发并与系统集成	30 人	4∶1	7
政府部门应用开发(内部)	30 人	5∶1	6
公司应用开发(内部)	30 人	4∶1	7

② 百分比法。根据测试人员应该占到项目组中人员的百分比数量。表 13-3 是一个百分比法的示例。

表 13-3　百分比法示例

开发类型	项目人员数量	测试组规模比例	测试组规模
商业产品(大型市场)	50 人	27%	13
商业产品(小型市场)	50 人	16%	8
单个客户端的应用开发	50 人	10%	5
单个客户端开发并与系统集成	50 人	14%	7
政府部门应用开发(内部)	50 人	11%	5
公司应用开发(内部)	50 人	14%	7

③ 测试程序法。根据测试程序数量,以及每个程序可能的执行时间,计算出人时(1 人工作 1 小时称为 1 人时),再根据完成周期计算测试组规模。表 13-4 是一个测试程序法的示例。

表 13-4　测试程序法示例

	测试过程数目	计算因子	人小时	完成周期	测试组规模
历史记录	860	6.16	5300	9 个月	3.4
新项目评估	1120	6.16	6900	12 个月	3.3

④ 任务计划法。根据历史记录中类似项目工作量,比较新项目同历史项目的工作量,历史项目乘以相应的因子。先将任务分解,根据历史记录乘以一个因子,计算出新项目的全部工作量,再根据该工作量和完成周期计算测试组规模。

13.1.2 测试过程组织

测试过程组织主要包括测试过程规划、测试实施、过程改进。

1. 测试过程规划

由一位对整个系统设计熟悉的设计人员编写测试大纲，明确测试的内容和测试通过的准则，设计完整合理的测试用例，以便系统实现后进行全面测试。为了保证测试的质量，可以将测试过程分成几个阶段，如代码审查、单元测试、集成测试和验收测试。可以根据实际情况来确定适合自己的测试过程。

2. 测试实施

① 组织测试人员仔细阅读有关资料，包括规格说明、设计文档、使用说明书及在设计过程中形成的测试大纲、测试内容及测试的通过准则，全面熟悉系统，编写测试计划，设计测试用例，做好测试前的准备工作。

② 组织技术人员根据软件项目测试的要求，搭建测试环境。搭建测试环境时要时刻保持测试环境中配置参数的正确、软件版本的持续更新。好的做法是专人专岗，负责维护测试环境，解决测试中由测试环境引起的问题。

③ 明确测试任务与职责分配。根据测试软件项目的规模、完成时间，确定参与测试的人数，测试负责人把测试任务分配给每一个测试人员，明确每一个测试人员的测试内容及要求。

④ 组织测试文档评审。测试人员按照要求撰写有关的测试文档，提交公司测试管理人员，管理人员召开测试评审会议对测试文档进行评审。

⑤ 组织测试协调会。协调在一个大型应用系统中非常重要，渠道畅通，流程环节少，解决问题就快，否则，问题积累得就会越来越多，影响整个系统的测试。召开协调会的目的既是对前一段工作进行总结，也是对工作中存在的问题商讨解决的方法。测试协调会包括测试前会议、测试过程中会议、测试总结会等。

3. 过程改进

俗话说过程决定结果，测试过程的质量决定了软件测试的质量和效果。一个好的测试管理者应该有过程改进的勇气，并有矢志不渝的决心。因为，只有不断地改进才能使测试过程更加合理、测试方法更加科学有效，不断地引领测试组织向更成熟的方向迈进。测试人员在这样的组织里会更加具有归属感，在一个不断成长的组织里，个人技能也会获得更多提升的机会。这样的组织才富有长久的生命力。

13.2 软件测试管理

软件测试生命周期向我们展示了软件测试的全过程，在这个过程中我们要制定测试计划，进行测试设计和开发编写测试脚本，执行测试用例，记录所发现的缺陷并对缺陷进行跟踪和管理。软件测试过程中的这几个阶段环环相扣，关系密切，因此科学地管理和组

织好这个过程是保证软件测试质量的必要条件。

测试管理的目的是提高测试效率,保证软件测试的质量。归纳起来,软件测试管理的内容有以下三个方面,它们是测试过程管理、配置管理、风险管理,涵盖了人员管理、设备管理、测试进度管理、测试用例管理、缺陷管理、过程记录及文档管理、测试成本管理以及风险管理等软件测试过程的方方面面。其中有些内容,我们在讨论软件测试的一般过程和方法时已经做了阐述,对于这部分内容本节则不再赘述,比如缺陷管理等。

13.2.1 测试过程管理

测试过程管理主要对测试过程中的测试活动和测试资源进行管理,主要是测量和分析软件测试过程的有效性和效率。

1. 测试活动管理

软件测试活动主要包括制定测试计划、测试设计和开发、执行测试、评估测试等,对这些活动的管理主要是对各个活动进行监督、度量、控制,保证各个活动的顺利有效进行,管理内容主要包括测试设计管理、测试进度管理、测试成本管理等。

1）测试设计管理

测试设计主要包括测试计划、测试方案、测试用例等的设计。测试设计的质量决定了测试的覆盖率,对软件产品的最终质量有很大影响,所以对测试设计的管理显得非常重要。如何通过管理来保证测试设计的质量,是每个测试管理人员需要认真考虑的事情。为了保证有关测试设计的质量,可以采用如下方式来进行。

(1) 加强理解。对软件产品的理解程度,决定着测试设计的质量。可以参考如下几种方式进行:

- 让测试设计人员加入开发测试团队,深入了解开发需求;
- 建立有效的沟通机制,促进测试人员与软件使用人员、软件设计人员、软件开发人员等的沟通、交流;
- 加强专业培训,让设计和开发人员为测试人员做专项内容介绍;
- 让测试人员讲解自己对软件产品特性和功能的理解。

(2) 责任到人。将软件设计任务划分清楚,责任到人,提高每个测试用例设计人员的责任感。每一个测试模块都有专人负责,从需求分析开始到测试设计,确保测试设计的质量。负责某个模块的测试人员要和开发人员一起工作,以便于对产品特性设计进行充分讨论,了解其实现的机理。

(3) 加强评审。测试设计文档编写完毕后,需要进行相应的评审。测试文档评审的目的是保证各阶段测试文档格式、内容等方面符合项目测试要求,保证测试设计的质量。

评审方式可以通过召开测试评审会议进行评审,可以采用同行评审。另外可以通过检查单的方式对测试设计文档进行审查。

检查单是软件质量管理活动中最常用的工具之一,通过检查单提醒检查人员检查哪些内容,避免遗漏。以下为测试用例检查单的部分示例内容:

- 是否涵盖了需求文档上的每个功能点。
- 是否涵盖了需求文档上的每条业务规则说明。
- 是否覆盖了输入条件的各种有意义组合。
- 是否覆盖了业务操作的基本路径和异常路径。
- 是否考虑了重要表单字段的数据合法性检查。
- 是否考虑了其他的测试类型(对某个功能很重要,但未在需求文档中提及的,如安全测试、周期性测试和故障恢复等方面)。
- 是否考虑了对其他模块/功能的影响。
- 是否使用了项目组的标准用例模板。
- 用例是否覆盖了测试设计中定义的所有场景。
- 用例编号是否统一、规范。
- 用例名称是否简洁、明了。
- 目的字段是否准确地描述了对应场景的测试输入的特征(不同数据、操作、配置等)。
- 前提条件字段的条目是否充分、准确,操作上是否不依赖于同组之外的其他用例。
- 对应的需求编号字段是否填写正确。
- 用例粒度、预估出的执行时间是否适当。
- 同组用例中,仅数据不同的,是否实现了测试步骤的重用。
- 某个功能点的第一个用例是否是基本流的。
- 操作步骤的描述,是否清晰、易懂。
- 操作步骤是否充分和必要,并具有可操作性。
- 测试用例的检查点是否明确、充分和可操作。
- 单个用例步骤或检查点中是否不再存在分支。
- 测试数据的特征描述是否准确,有条件的情况下,是否给出了一个当前环境下的可用参考值文字、语法是否准确;布局、格式是否统一。

④ 制定测试设计文档模版

按照 CMMI 软件能力成熟度模型原则要求,根据公司软件测试实际需要,建立相应的测试文档标准模板。在实际测试过程中,不论是对大型软件项目测试,还是小型软件项目测试,测试人员都要严格按照标准化的文档要求认真填写相关内容,力求做到语言简洁明了、内容准确无误、含义表达清楚。

2) 测试进度管理

软件测试项目的进度是国内企业普遍重视的项目要素。如何保证测试进度,是每个测试管理者关心的问题。

(1) 建立里程碑。一个项目往往是由若干个相对独立的任务链条组成的,各链条之间的协作配合就直接关系到整个项目的进度。这里可以用到著名的“木桶理论”,即进度最慢的项目就会是整个项目进度的代表。只有在任何一条链都已经优化的基础上,才可能进行系统的优化。因此,保证每条任务链的效率是整个项目进度优化的前提和基础。

通常,可以采用设置“里程碑”的方法来保证单独任务链的最优。

所谓里程碑,是指完成阶段性工作的标志,项目的不同阶段里程碑不同(如图 13-1)。设定里程碑的目的就在于将一个过程性的任务用一个结论性的标志标识,从而使得任务拥有明确的起止点。

由图 13-1 可以看到,一个测试计划任务由一个或多个测试里程碑组成,一个里程碑又可以进一步分解为一个或多个子里程碑。为了完成一个测试项目,我们需要设置一个或多个里程碑,用来标识测试过程中的预期结果和中间工作结果。

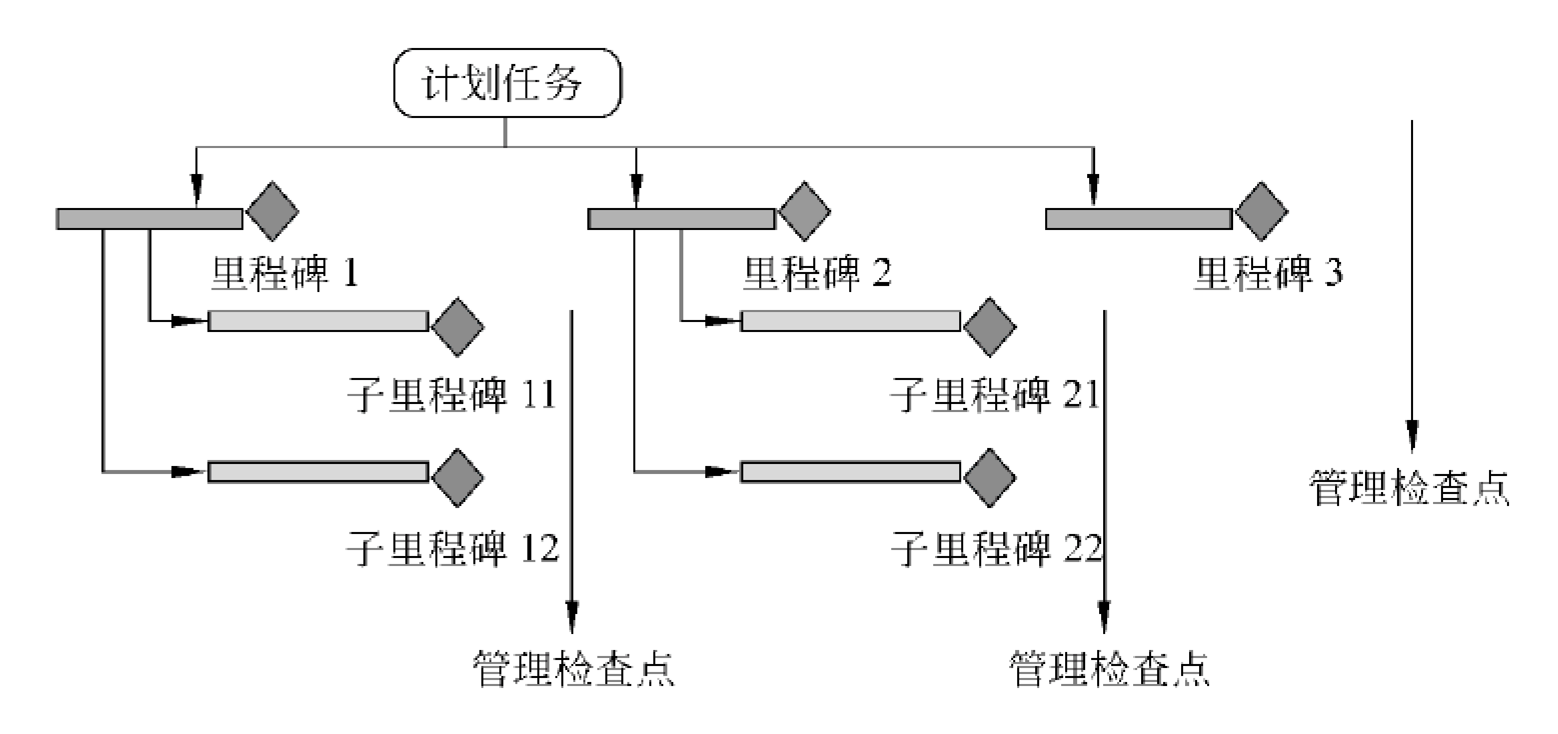


图 13-1 里程碑关系图

里程碑在项目管理中具有重要意义,在项目管理进度跟踪的过程中,给予里程碑足够的重视,往往可以起到事半功倍的效用,只要能保证里程碑的按时完成,整个项目的进度也就有了保障。里程碑的设置,通常以一些标志过程目标实现的重要事件体现出来。

里程碑设定了之后,为了检查和监督该里程碑的执行情况,我们可以为每一个里程碑设置一个管理检查点。通过管理检查点,我们可以看到每一个里程碑的时间计划、完成情况和资源分配情况。

管理检查点是针对每一个里程碑(包括子里程碑)而言的,有几个里程碑就有几个管理检查点,也就是说我们通过管理检查点来判断设立的里程碑是否已经达到了。管理检查点详细地描述了它对应的测试里程碑实施计划。

(2) 几点管理原则。

- 关注薄弱环节,实现动态平衡:测试的进度管理并不是一个静态的过程,测试的实施与测试的计划也是互动的。在测试进度的管理过程中,需要不断调度、协调,保证项目的均衡发展,实现项目整体的动态平衡。
- 适时督导,掌控进度:测试管理人员需要时时全面地了解每个测试人员的工作进展情况,可以通过每天或某一固定时间开总结会的方式,结合每个测试人员的工作产品情况来了解。测试人员在自己测试工作中遇到突发问题,或者测试难题等情况,可能会导致无法按时完成工作任务,要求测试人员提前上报测试管理者,以便测试管理人员根据情况提前想办法解决面临的问题,保证整个测试项目的进度。
- 前紧后松原则:对于接手的任务,要做到“前紧后松,赶早不赶晚”,尽力地按时、

甚至提前完成。

- 重者为先的原则：各种事务“按类别”、“分优先级”处理。分清轻重缓急，重者为先。bug 的处理也以功能性错误、死机死锁、致命等优先级为高，边缘死角问题为低的原则。

3）测试成本管理

测试成本也是测试管理者十分关注的一方面内容。测试成本一般包括：

- 人力资源成本：与项目人员相关的成本开销，包括项目成员工薪和红利、外包合同人员和临时雇员薪金、加班工资等；
- 资产类成本：资产购置成本，指产生或形成项目交付物所用到的有形资产，包括计算机硬件、软件、外部设备、网络设施、电信设备、安装工具等；
- 管理费用：用于项目环境维护，确保项目完工所支出的成本，包括办公室供应、房屋（租金，设备）、支持服务等；
- 项目特别费用：在项目实施以及完工过程中的成本支出，包括差旅费、餐费、会议费、印刷及复印等费用。

如何控制测试成本，是每个测试管理者不得不考虑的事情。控制测试成本主要从降低测试实施成本和降低测试维护成本考虑。

① 降低测试实施成本。准备测试环境时，应将测试环境建立在固定的独立的测试专用软、硬件及网络环境中，使测试工作能够独立开展，减少外部干扰。

自动化测试可以加快测试执行时间，是降低测试实施成本的一个不错的选择。测试管理者可以根据实际情况，有计划地开展自动化测试，积极引入适用的自动化测试工具，减少手工测试，提高测试效率。

② 降低测试维护成本。加强软件测试的配置管理，所有的测试样品、测试文档都应置于配置管理系统控制下，通过配置管理加强因频繁变更引起的信息共享，减少信息丢失现象的发生，使每个测试人员都能保证获取到测试产出物的最新版本，确保其所执行的测试是有效的。

2．测试资源管理

测试资源管理主要是对人力资源、工作环境、使用设备、软件资源的管理。

测试管理人员应该提供必要的软件、硬件资源，提供必要的基础设施，创造良好的工作环境。

人力资源是测试中的核心资源，人员资源管理主要包括测试人员的培训，测试人员的激励及绩效考评等。

1）人员培训

测试人员需要各方面的知识和技能，需要做好测试人员的培训学习。不管是公司内部自行测试，还是外包测试，测试人员自身业务水平的高低和对系统掌握的熟练程度，都直接关系到系统测试的质量和工作进度。为此，在正式对软件测试前，应根据项目要求对测试人员进行集中培训，通过培训让测试人员熟练掌握系统的结构和功能，必要的测试方法和测试管理流程等，对公司整个软件应用系统及测试管理制度及流程有一个完整的认识。

2）人员考核

人员考核可以从工作质量、工作进度、工作态度等几个方面来进行考核。建立有效的机制，使测试人员能够达到测试计划规定的工作目标，并收集和分析软件人员的工作绩效数据等方面的内容。

（1）工作质量。考核测试人员工作的质量，可以从用例质量、文档质量、系统质量等方面进行考核。

- 用例质量：可以用总有效缺陷除以用例总数，得出单位用例的缺陷检测率，用以考核用例设计的质量；
- 文档质量：主要体现在测试计划、方案、评估报告的考核上，主要从规范性、及时性、有效性等方面进行评估；
- 系统质量：主要考察有效缺陷质量、缺陷泄露率、有效缺陷比、各级别缺陷比重等指标。

（2）工作进度。主要考核测试人员整体工作效率，也是人员能力体现的一个方面。主要指标包括设计效率、设计覆盖率、执行效率、执行覆盖率、是否在指定时间内完成工作任务（主要考察实际和计划的偏离度），对于以上单项指标，根据团队规模按照一定比例划分，如前 10％分值为 5 分，中间 80％分值区间是 2～4 分，后 10％为 0～2 分等。

（3）工作态度。考核测试人员在测试过程中的责任心、主动性方面，考察测试人员对待工作是否认真负责，以及主动思考、解决测试中可能出现的各种问题的能力等。

考核方式可以采用管理人员直接对测试人员进行考核的方式，也可以进行项目组内成员对其他人员评价的方式，可以根据客户对系统的反馈情况，对相应的测试人员进行考核，或者几种考核方式综合来进行考虑。

13.2.2　配置管理

随着软件系统的日益复杂化和用户需求、软件更新的频繁化，配置管理逐渐成为软件生命周期中的重要控制过程。随着软件测试的发展，配置管理被引入到软件测试过程，并且扮演着越来越重要的角色。

配置管理主要是对测试过程产品进行标识、存储和控制，以维护其完整性、可追溯性以及正确性。配置管理的基本单位是配置项，配置项可以为测试计划、测试说明书、测试用例、测试报告、缺陷报告等。配置项可以是文件级粒度的，也可以是文件版本级粒度的。当然，粒度越小管理的成本越高，但是配置的精度也就越高。一个完整的配置管理系统要具有配置标识、版本控制、变更控制、配置状态统计和配置审核功能。其中变更控制包括基线管理、变更请求管理、构建管理和发布管理（如图 13-2 所示）。

1. 配置标识

配置标识就是识别产品的结构、产品的构件及其类型，为其分配唯一的标识符，也就是说，每一个配置项要有一个唯一标识。一般来说，标识包括两个方面：一是文件名，二是版本，可用如下一个二元组来标识：＜文件名，版本＞。每个项目首先要确定一套命名规则，例如，某系统测试用例采用“系统. 子系统. 模块. 用例文件”的方式来建立规则标识。

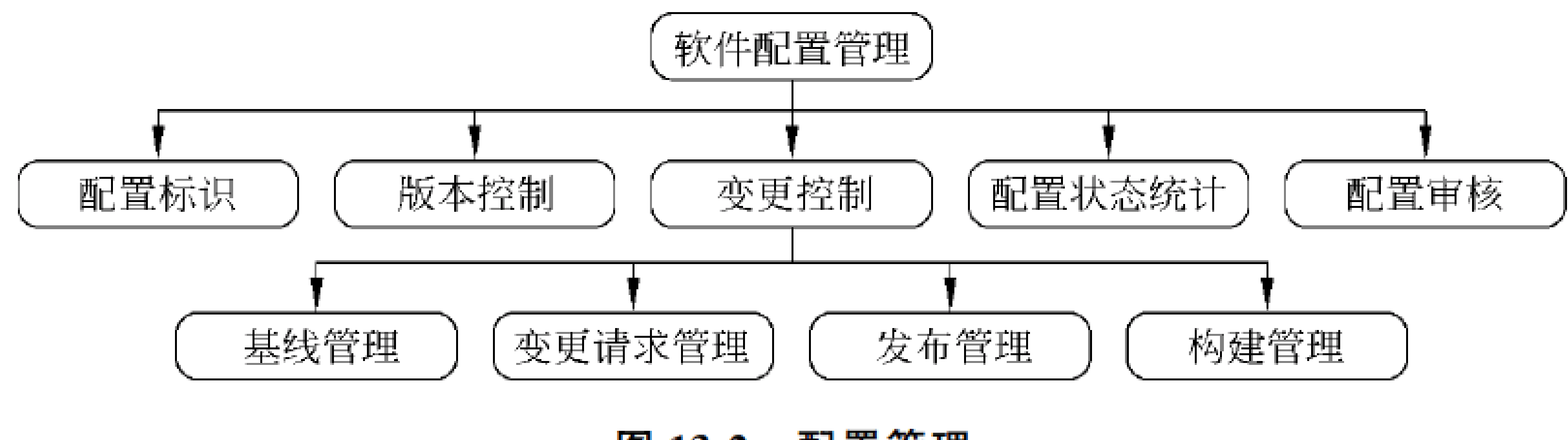


图 13-2　配置管理

2. 版本控制

版本控制就是对在软件测试过程中所创建的配置对象的不同版本进行管理，保证任何时候都能取到正确的版本以及版本的组合。当前，这方面典型的工具有如 VSS、CVS 和 SVN。

3. 变更控制

软件测试过程与软件开发过程一样，存在着许多变更，比如，测试计划的延期、测试用例的更改和废弃、缺陷状态的变更等。对于这种变更，要建立一个控制机制，以保证所有变更都是可控的、可跟踪的、可重现的。对变更进行控制的机构称为变更控制委员会(Change Control Board，CCB)，变更控制委员会要定期召开会议，遵循一定的变更机制，对近期所产生的变更请求进行分析、整理，并做出决定。

4. 变更请求管理

变更请求管理就是对变更请求(Change Request，CR)进行分类、追踪和管理的过程来实现的。对变更请求的有效管理可以提高产品管理的透明度，有关人员通过这种管理机制可以清楚地知道当前产品的进展情况，比如有多少个新产生的 CR，已经解决了多少 CR 等等，有利于有关人员做出正确的决策。

5. 基线管理

基线是指经过正式评审和批准，可作为下一步工作基准的一个配置。通过基线管理可以使用户能够通过对适当版本的选择，来组成特定属性(配置)的软件系统，这种灵活的“组装”策略使得配置管理系统像搭积木似的使用已有的积木(版本)组装成各种各样、不同功能的模型。这是在开发领域基线管理的好处，在软件测试领域，基线管理为测试人员提供了这样一种工作模式：不同测试人员各自完成一个大型系统的各个子系统的用例分析和设计工作，阶段性成果完成后标识为一个基线，测试负责人或低一级的测试小组负责人可以提取这些阶段性基线，合并后形成整个系统或者一个稍大规模的子系统的测试用例集。

基线的变更需要一个严格的流程，需要提出申请，经过审批，然后才能进行。

6. 构建管理

在做构建时，需要首先取出正确的配置，然后再做构建。可以利用基线，可以取出某个基线的所有配置项，也可以利用配置管理系统的构建功能直接在工作空间内做构建。构建管理需要配置管理工具的支持。

7. 状态报告

状态报告要回答所谓 4W 的问题：

- What：发生了什么事？
- Who：谁做的此事？
- When：此事是什么时候发生的？
- Why：为什么做此事？

状态报告要能够报告所有配置项以及变更请求的状态，通过量化的数据和报表反映项目开发和测试进度的状态。

8. 配置审核

配置审核要审查整个配置管理过程是否符合规范，配置项是否与需求一致，记录正确，配置的组成是否具有一致性等。

13.2.3 风险管理

软件测试风险是不可避免的、总是存在的，所以对测试风险的管理非常重要，必须尽力降低测试中所存在的风险，最大程度地保证测试工作的正常进行。在此，主要介绍软件测试过程中常见的测试风险和人员风险等。

1. 测试风险

在测试工作中，主要的风险有：

① 质量需求或产品的特性理解不准确，造成测试范围分析的误差，结果某些地方始终测试不到或验证的标准不对。

② 测试用例没有得到百分之百的执行，如有些测试用例被有意或无意的遗漏。

③ 需求的临时/突然变化，导致设计的修改和代码的重写，测试时间不够。

④ 质量标准不够清晰，如适用性的测试，仁者见仁、智者见智。

⑤ 测试用例设计不到位，忽视了一些边界条件、深层次的逻辑、用户场景等。

⑥ 测试环境，一般不可能和实际运行环境完全一致，造成测试结果的误差。

⑦ 有些缺陷出现频率不是百分之百，不容易被发现；如果代码质量差，软件缺陷很多，被漏检的缺陷可能性就大。

⑧ 回归测试一般不运行全部测试用例，是有选择性的执行，必然带来风险。

有的风险是可以避免的，如上述的风险①～③；而有的风险是不能避免的，如上述的风险④～⑦，这些不可避免的风险我们可以尽可能地把这类风险的影响降到最低；有的测试风险虽然可以避免，但是出于时间或成本的考虑而不特别采取规避措施，如上述风险的最后一种回归测试风险。

针对上述软件测试的风险，有一些有效的测试风险控制方法，例如：

- 测试环境不符，可以通过事先列出要检查的所有条目，在测试环境设置好后，由其他人员按已列出条目逐条检查；
- 有些测试风险可能带来的后果非常严重，能否将它转化为其他一些不会引起严重后果的低风险。如产品发布前夕，在某个不是很重要的新功能上发现一个严重的

缺陷，如果修正这个缺陷，很有可能引起某个原有功能上的缺陷。这时处理这个缺陷所带来的风险就很大，对策是去掉（Diasble）那个新功能，转移这种风险；

- 有些风险不可避免，就设法降低风险，如“程序中未发现的缺陷”这种风险总是存在，我们就要通过提高测试用例的覆盖率（如达到99.9%）来降低这种风险。

为了避免、转移或降低风险，事先要做好风险管理计划和控制风险的策略，并对风险的处理制定一些应急的、有效的处理方案，例如：

- 在做资源、时间、成本等估算时，要留有余地，不要用到100%；
- 在项目开始前，把一些环节或边界上的可能会有变化、难以控制的因素列入风险管理计划中；
- 对每个关键性技术注意培养后备人员，作好人员流动的准备，采取一些措施确保人员一旦离开公司，项目不会受到严重影响，仍可以继续下去；
- 制定文档标准，并建立一种机制，保证文档及时产生；
- 对所有工作多进行互相审查，及时发现问题，包括对不同的测试人员在不同的测试模块上相互调换；
- 对所有过程进行日常跟踪，及时发现风险出现的征兆，避免风险。

要想做好风险管理工作，就必须彻底改变测试项目的管理方式，针对测试的各种风险，建立一种“防患于未然”或“以预防为主”的管理意识。与传统的软件测试相比，全过程测试管理方式不仅可以有效降低产品的质量风险，而且还可以提前对软件产品缺陷进行规避、缩短对缺陷的反馈周期和整个项目的测试周期。

2. 人员风险

在项目测试中，还应充分估计到人员风险。由于工作环境、待遇、工作强度、公司的整体工作安排和其他无法预知的因素，一个项目尤其是周期较长的项目几乎不可避免地要面临人员的流动。如果不在项目初期对可能出现的人员风险进行充分的估计，作必要的准备，一旦风险转化为现实，将有可能给整个项目造成很大的损失。及早预防是降低这种人员风险的基本策略，下面我们从第三方测试的角度具体介绍一下人员风险的控制方法：

- 保证团队中全职人员的比例，且项目核心部分的工作应该尽量由全职人员来担任，以减少兼职人员对项目组人员不稳定性的影响。
- 建立良好的文档管理机制，包括项目组进度文档、个人进度文档（测试日志）、版本控制文档、整体技术文档（测试策略、测试用例）、个人技术文档（测试执行记录、缺陷报告）等。一旦出现人员的变动，比如某个组员因病退出，替补的组员能够根据完整的文档尽早接手工作。
- 加强项目组内技术交流，比如定期开项目例会，使测试组成员能够相互熟悉对方的工作和进度，能够在必要的时候接替对方工作。
- 对于项目经理，可以从一开始就指派一个副经理或者项目经理助理协同项目经理管理项目开发工作，如果项目经理因故退出项目，副经理或者项目经理助理可以很快接手。但是，一般只建议在项目经理这种比较重要的岗位采用这种冗余复制的策略来预防人员风险，否则将大大增加项目成本。
- 为项目测试工作的开展提供尽可能好的基础环境，比如工作环境、待遇、工作进度

安排等，同时一个优秀的项目经理应该能够在项目组内营造一种良好的人际关系和工作氛围。良好的开发环境对于稳定项目组人员以及提高生产效率都有不可忽视的作用。

本章小结

软件测试组织与管理是软件测试领域一个重要的研究方向，本章借鉴软件开发领域一些不错的经验做法，试图为软件测试的组织与管理打开一个新的思维视角。

本章介绍的软件测试的组织主要分为两部分，一是测试人员的组织，如组织结构、人员组成、组织规模等几个方面；二是测试过程的组织，主要包括测试过程规划、测试实施、过程改进等。软件测试管理主要从过程管理、配置管理、风险管理等三个方面做了分类介绍。其中，过程管理主要对测试过程中的测试活动和测试资源进行管理；配置管理主要对测试过程中产生的各种工作产品进行管理，测试工作产品包括测试计划、测试说明书、测试用例、测试报告、缺陷报告等；风险管理主要是对测试过程中的风险进行识别和控制，并制定针对性的措施，防止风险发生，或者把风险降到最小。

CHAPTER

第 14 章 测试度量与过程改进

人们对软件测试的重视程度越来越高，在软件产品开发中测试的比重也越来越大。但是随着软件产品开发的复杂程度的提高，传统的软件测试流程不可避免会产生以下一些问题：

- 手工操作过多，导致测试进展缓慢，以及人力成本居高不下；
- 测试在开发基本完成才启动；
- 测试用例的重用率低，导致回归测试时难以做到重复利用；
- 测试人员为新招收的技术人员，并且没有相应的培训机制，测试工作效率低；
- 测试进度超过预期。

以上问题基本上是一般软件测试项目中普遍存在的，为了能适应软件测试的需要，需要对原有的测试过程进行相应的改进，以使测试达到更好的效果，保证软件产品的质量。

要改进软件过程首先要了解测试过程目前的状况，需要收集测试过程中的进度、成本、资源和质量等数据信息，并对这些信息进行分析。而收集测试过程信息及分析，则是软件测试度量所包含的内容。基于度量与分析的可持续过程改进方法，在工业界和学术界已经取得了共识。组织可以自定义需要度量的过程数据，将收集来的数据加以分析，以找出需要改进的因素。在不断的改进中，同步的调整需要度量的过程数据，使度量与分析始终为了过程改进服务，而过程改进也包含对度量和分析的改进。掌握了基于度量和分析的可持续过程改进方法，测试过程管理将得到不断完善，测试活动将能够始终处于优化状态。

14.1 测试度量

14.1.1 什么是度量

软件测试过程度量是对软件测试过程的量化分析，它提取软件测试过程中可计量的属性，在测试过程进行中以一定频度不断地采集这些属性的值，并采用一些恰当的分析方

法对得到的这些数据进行分析，从而量化地评定测试过程的能力和性能，提高测试过程的可视性，帮助软件组织管理以及改进软件测试过程。

软件测试度量的目的是用软件度量学的方法来科学地评价软件测试质量，控制和管理软件测试过程，合理组织和分配资源，以较低的成本获得高质量的软件。软件测试度量为软件测试过程的不断改进和量化管理奠定了基础，也为测试管理人员了解项目当前所处状态(如进度情况、重要任务完成情况、需求变更情况等)提供了帮助。

14.1.2 测试度量内容

测试度量是用来分析软件测试的状况，在软件测试中，需要测量的度量内容主要包括进度、成本、质量等几个方面。

1. 进度度量

- 计划的测试开始、结束时间；
- 实际的测试开始、结束时间；
- 执行测试用例的时间。

2. 成本度量

- 计划投入测试的工作量(人时)；
- 计划投入测试的资金；
- 实际投入测试的工作量(人时)；
- 实际投入测试的资金；
- 评审投入的工作量(人时)；
- 缺陷修正成本；
- 累积测试时间。

3. 测试质量度量

- 需求覆盖率；
- 测试用例覆盖率；
- 测试用例执行率；
- 测试用例通过率；
- 缺陷检测率。

4. 产品质量度量

- 版本发布前缺陷数；
- 版本发布后缺陷数；
- 评审发现的缺陷数；
- 缺陷修正率；
- 缺陷密度。

14.1.3 测试度量分类

对软件测试过程的度量是对软件测试过程特性的一种描述，不同的描述类型决定了

不同的度量内容、度量的表达方式、度量数据的采集手段、度量数据的分析方法。根据不同的描述类型划分测试过程度量类型如下。

1. 客观度量和主观度量

客观度量是过程或产品的实际结果，主观度量是人的主观判断结果，也可以是在客观数据基础上的分析结果。实施主观度量的一个前提是要有经验丰富的软件工程和软件过程人员。客观度量在一定程度上减少了人为的主观影响，因为它多表现为量化的数据，且数据的来源和产生的背景信息又可以清楚定义，这就赋予了数据较强的说明能力，为软件组织对过程进行量化管理和优化管理奠定基础。

2. 绝对度量和相对度量

绝对度量是指其度量值的取得没有参照物或没有其他属性之间的依赖关系，相对度量是指其度量值的取得具有参照物或与其他属性之间有依赖关系，比如生产效率依赖于过程时间和产品的规模。

3. 显式度量和隐式度量

显式度量是可直接得到数据的度量，隐式度量是对原始度量数据进行运算或结合多个度量分析得到的结果。例如测试时间是显式度量，测试人员工作效率是隐式度量。

4. 动态度量和静态度量

动态度量是二维以上的度量，用于动态监控过程信息随时间的更新情况。一般第一维表示数值或类别，第二维表示时间。静态度量是一维度量，例如过程的计划信息。

5. 预测度量和解释度量

预测度量是根据即有度量信息来预测过程的未来执行情况。解释度量是事后度量，对已经执行的活动进行度量。从过程迭代的角度来看，上一次的解释度量可能是下一次的预测度量。

6. 内部度量和外部度量

内部度量和外部度量的划分是从度量的信息源和结果应用的作用域而言的。

14.1.4 测试度量过程

测试度量过程主要包括确定度量信息需求、制定度量计划、执行度量、评估度量。

1. 确定度量信息需求

在进行测试度量时，首先要确定软件测试度量的信息需求，即确定要解决的问题是什么；然后采取自顶向下逐步细化的策略，分析软件测试度量的信息需要。主要从以下几个方面来分析提取测试度量信息需求：

- 软件测试策略方面，主要是采用的测试技术、测试方法等；
- 测试技术方面，主要是解决测试的预算、测试的工作量、测试成本的估算、可以使用的时间、已用的时间等；
- 测试过程中间产品方面，主要解决测试用例的设计的效率、平均生产成本、测试用

例平均发现缺陷的个数、测试用例的总个数，脚本的设计的效率、平均生产成本、脚本平均发现缺陷的个数、脚本的总个数，缺陷的平均发现成本、工作量等；

- 测试过程有效性方面，主要是解决测试的缺陷发现率、测试用例和脚本的复用率、回归测试的工作量、测试计划的实施效果等；
- 测试管理和控制方面，主要是控制测试的进度，控制测试的成本，配置合适的测试人员等方面。

2. 制定度量计划

在测试度量信息需求确定的基础上设计测试度量计划，在设计度量计划中要做的主要工作有确定度量策略、构造度量构造、设计基本度量、设计派生度量、设计指示器等。度量构造主要融合在基本度量、派生度量和指示器中，通过设计这些相关的度量来构造度量构造。

软件测试基本度量，是由一个指定的度量方法定义的对软件测试中单个属性的度量，执行该度量将产生一个度量的值。一个软件测试的基本度量在功能上是独立于其他所有度量，并获取单个属性的信息。

软件测试派生度量是一种基于基本度量的度量，它被定义成两个或多个基本度量或派生度量的一个函数，派生度量获取多于一个属性的信息。

软件测试度量指示器是提供指定属性的估计或评价的度量，该属性是从涉及已定义的信息需求的分析模型中派生。指示器一般包括度量元、决策准则。

3. 执行度量

执行测试度量包括收集和处理度量数据。使用数据来分析信息需求以及信息需求和相关问题的内在联系，生成信息产品以提供分析结果、候选的行动方案以及可供项目决策者参考的建议，控制测试过程。本阶段负责收集各种数据资源为分析数据做准备，并将数据存储在分析模型能够被访问的地方。

然后执行度量计划，实施度量进行数据收集、分析处理，最后通过指示器的分析得出信息产品来对决策提供支持；通过评估测试过程来改进测试过程、加强测试的过程管理、提高测试效率，最终解决度量初始阶段提出的问题完成度量目标。

4. 评估度量

测试度量的评估是将度量和分析技术应用到度量过程本身。它评估应用的度量和度量过程的能力，帮助标识相关的改进措施。评价度量活动确保项目度量方法的持续更新，以满足当前的信息需求，并促进项目及组织级度量过程的成熟度的提高。

14.2 测试过程改进

14.2.1 测试过程改进内容

测试常被看做是一个昂贵且不可控的过程，因为传统上测试往往要花费很多的时间，耗费的比计划投入的多，受各种因素的影响已定义的测试过程总是被不断地扭曲，导致无

法度量和提供其质量情况。在实际工作中,测试人员常常被问到的一个问题就是“测得怎么样了?”,测的怎么样了,就是这样一个简单的问题却难倒了一众测试同行。这是因为测试过程是一个既有管理、也有技术的测试实施活动,哪一方面出现问题或者规划不合理都会降低测试的实效,比如在技术方面的测试方法、测试用例的设计等。测试往往表现出这样一个特点,即只要测,就总会有问题被不断地发现,这是个令人稍感沮丧的问题。如果困惑于这一点,那么我想一定是忘记了软件测试中一个重要的原则:穷尽测试是不可能的,因此适时地结束软件测试成了每一个测试管理人员的一个很重要的职责。但是,要做到“适时”绝不是一件容易的事情,这需要测试管理人员对测试过程有一个很强的把控能力,要力争做到既能降低测试成本,同时又能保证测试的有效性,而且要求对于不同的项目也要能很好的适应。事实上,这些正是测试过程改进的一个主要目标。

测试过程存在周期长,测试内容多的特点,要保证测试过程有效需要进行科学的管理,如测试进度的控制、测试资源的管理、测试用例的管理、测试缺陷的管理、测试的配置管理等。这些管理内容和管理方法并不是一成不变的,因此我们要想长期保持管理的有效性,就需要不断地对我们的管理进行改进。

软件测试也是一项脑力密集性劳动,从业人员的素质对软件测试的质量有着直接影响,显然人力资源的持续开发也是过程改进的一项不可分割的内容。

管理、技术、人员构成了测试过程改进的三个基点,下面将从这三个方面论述一下测试过程改进的有关内容和实践。

1. 测试管理改进

1) 进度管理

进度管理的好坏直接影响整个测试工作的质量,所以在软件测试过程中,对于进度的控制历来受到重视。如果软件测试进度出现延期现象,需要改进进度管理方式。进度管理改进主要是定义明确的测试过程,提高测试计划的可执行性,合理的安排测试活动顺序等。

(1) 提高测试计划的可执行性。

计划的可执行性是软件工程领域里的一个顽疾,在软件测试领域同样存在这样的问题。很多时候,我们会看到这样的场景:项目测试初期,测试负责人会编制一份测试计划,这个是没有问题的,但是问题在于对这个计划的维护做得不够甚至没有,项目进行过程中不可避免地会发生很多种变更,比如用户需求变更、开发计划变更、人员变更等,这些变更必然会对测试计划造成冲击,使原计划工作内容无法再进行,这种情况下对测试计划的修改势在必行,否则测试计划就脱离了实际,可执行性大幅降低。不具备可执行性的测试计划显然会大大降低测试人员编制测试计划的热情,长期如此,也会影响到项目组的其他人员,使之逐渐丧失阅读测试计划的兴趣,对于测试计划的工作内容也不关心,更谈不上可能的支持。

从上面的分析可以看出,测试计划可执行性差的一个直接原因是计划与实际脱离,那么对症下药,我们在测试计划中需要采取的改进步骤就是要对测试计划给予足够的重视,对于项目过程中的变更要及时反映到测试计划中,保持一致性。

另外,从本书第三篇有关内容中可以知道测试计划包括测试方法、测试项通过准则、

测试任务、进度安排、环境要求、测试人员职责、测试风险等内容，测试人员在编制过程中对每部分内容都要清晰明确(如进度安排要尽量考虑周全，对测试任务要尽量细化，整理出每部分任务的准入、准出条件，及各任务之间的关系，并且责任明确到人)，要做好风险分析，每部分工作内容要留有一定余地，防止测试执行时因意外因素导致任务无法按照原计划完成，要有明确的里程碑及检查点以便于对计划内容的跟踪和考查。

(2) 合理安排测试活动顺序。

不合理的测试顺序，可能会引起测试效率低下。如果不立即采取相应的措施，会导致测试进度的失控。例如在一个项目中，哪些测试活动必须有先后顺序，哪些测试活动能够并行开展，哪些测试活动可以合并完成，哪些测试活动存在时间上的线序关系，一定要区分清楚，并做最优化调整。

2) 成本管理改进

如果测试中人工测试占用的比例较大，造成人力成本过高，可以考虑引入自动化测试，同时可以对测试人员进行相应的培训，提高其工作效率，节省人力成本。

3) 资源管理改进

资源管理改进主要是优化测试资源配置。测试活动涉及到人力、设备、场地、软件环境与经费等资源，如何合理地调配各项资源给相关测试活动也是非常值得斟酌的。最常见的就是人力资源的调配，测试部门如果能深入了解员工的专长与兴趣所在，能对测试活动的开展起到事半功倍的效果。

2. 测试技术改进

1) 重视测试需求分析

测试需求分析在测试过程中起着重要的作用，需求分析不准确、不透彻，会导致测试结果存在偏差，“差之毫厘，谬以千里”。在测试过程改进中要特别注意这一点，很多公司的测试人员都不太重视测试需求分析，由于时间紧或测试人员有限，不得不看了一部分需求，就开始编写相应的测试用例，导致测试需求不充分，甚至出现偏差，测试的效果不理想。

充分了解测试需求，了解获取需求的渠道，保证测试需求比较全面准确，同时可以列出功能测试或业务测试需求点，让用户和项目组成员评审确认，避免测试后期由于时间原因而导致漏测。

2) 确定合理的度量模型和标准

度量结果是测试结果评估的重要依据，所以度量模型的选择和度量标准的确定显得尤为重要。确定一个比较合理的测试度量标准，不断实践、不断总结、不断改进，提出符合公司实际情况的测试过程改进措施。一旦确立了质量度量模型和标准，很多测试活动就不至于陷入过度测试或测试不够的尴尬状态中。

3) 提高测试覆盖率

在保证测试成本的前提下，要尽可能提高测试的覆盖率。我们可以通过测试用例覆盖率来提高测试的质量，保证较多的缺陷在版本发布以前被测试组发现。也可以通过提高测试技术覆盖率，用实践来验证，不能够轻易相信部分行业人士的一面之词，而误入歧途。在保证公司盈利的情况下，尽量提高测试的覆盖率，提高测试质量。测试覆盖包括测

试内容覆盖、测试技术覆盖和测试过程覆盖。

提高内容覆盖：不论是起草测试计划、设计测试用例、执行测试用例还是跟踪软件缺陷，内容覆盖率越高，就越能避免故障被遗漏的情况。

提高技术覆盖：对于一项技术指标要尽可能地做到测试技术的覆盖，不必迷信某位专家或者专业人士，但必须相信他们提出的科学的验证方法，采用越科学的方法来验证某项指标，我们对产品的质量就越有信心。

提高测试过程覆盖：在测试过程中如果疏漏了一个重要环节，就不能很好地达到预先的目标。比如有些企业根本不写测试用例就让测试人员去做测试；有些企业根本不做测试用例评审，就匆匆发布出去让测试人员执行测试用例；有些企业测试人员不知道测试的标准，就去报 bug。这些看似不很重要的工作流程一旦被省略往往就造成工作无所适从，难于开展，成为测试活动失败的关键原因所在。

4）引入自动化测试

自动化测试能够替代部分手工测试工作，避免机械地重复测试，同时，它还能够完成手工无法完成的性能测试工作，如并发用户测试、大数据量测试、长时间运行可靠性测试等。自动化测试具有如下优点：

- 提高测试质量：软件开发的过程就是一个持续集成和改进的过程，而每一次修改都有可能产生错误。因此，当软件产品的一部分，或者全部，或者应用环境被修改时都需要对软件产品重新进行测试，其目的是验证修改后的系统或者产品的质量是否符合规格说明。例如，对于产品型的软件，每发布一个新的版本，其中大部分功能和界面都和上一个版本完全相似或完全相同，这部分功能特别适合采用自动化测试，由于自动化测试工具提供了简便的回归测试，能以便利的方式验证是否有新的错误引入软件产品，既节省了重复手工输入的工作量，保证了测试案例的一致性，避免了人为因素引入软件测试过程的缺陷，从而提高软件测试质量。
- 提高测试效率，缩短测试工作时间：软件系统的规模越来越大，功能点越来越多，达到几千个上万个，人工测试非常耗时和繁琐，这样必然会导致测试效率低下，而自动化测试工具可以较好地执行这些频繁的测试任务。在充分并合理地使用了测试工具以后，可以减轻测试工程师的手工测试工作，同时，测试工具还可以把控制和管理引入整个测试过程，能够保证测试的进度。
- 提高测试覆盖率：通过自动化测试工具的录制回放及数据驱动来测试功能，可以提高测试覆盖率。通过测试工具的辅助分析功能，可以提高测试的深度。
- 执行手工测试不能完成的测试任务：有些非功能性方面的测试，例如，压力测试、负载测试、大数据量测试、崩溃性测试等，人工测试是不可能实现的。
- 更好地重现软件缺陷的能力：自动化工具具有更好的一致性和可重复性，由于每次自动化测试运行的脚本是相同的，所以每次执行的测试具有一致性，人是很难做到的。
- 更好地利用资源：理想的自动化测试能够按计划完全自动地运行，在开发人员和测试人员不可能实行三班倒的情况下，自动化测试可以胜任这个任务，例如，完全可以在周末或者晚上执行测试。这样充分地利用资源，也避免了开发和测试之间

的冲突。

5）优化测试文档设计

测试文档是项目测试过程中测试人员之间交流的一种无声的语言，它记录了项目测试过程中有关测试配置、测试运行、测试结果等方面的信息，有利于项目管理人员、测试人员之间的交流和合作。文档编写不清晰、不规范，会引起测试人员交流的不充分，甚至引起误解，影响测试的效率。

可以制定相应的文档模版来优化测试文档，提高文档的可读性。如测试计划模版、测试用例设计模版、测试过程记录模版、测试报告模版等。

3. 测试人员

测试人员方面的改进主要是根据组织的情况，适度招聘或减少一定数量的测试人员，以适应组织当前阶段的工作任务要求，规划组织结构，进行岗位定义和职责划分，使专人专岗，各司其职。另外，根据组织发展情况和员工技能评估情况进行有组织有目的的相关培训，包括知识方面的培训、技能方面的培训等，使员工能够适应组织未来的发展。

例如，安排专门的人员进行配置管理，专门的人员进行资源管理等；对测试人员进行测试技术的培训、测试工具使用的培训等。

14.2.2 测试过程改进过程

软件测试过程改进是一个循序渐进的过程，且过程改进的效果通常需要一定的时间和数据来证明，需要经过一系列的实施步骤来进行：

- 确定测试过程改进目标。
- 制定软件测试过程改进计划。
- 建立跟踪控制机制：测试过程的改进需要建立相应跟踪，最好由专人来负责，定期定时定点输出相应记录信息。
- 实施测试过程改进策略：制定了测试过程改进计划，应去执行具体的流程操作，然后要注意评审和验证，定期定时定点监控，采集测试过程改进度量数据。
- 反馈总结再总结：总结测试实施过程中的经验，然后修改调整项目计划及实施改进的策略。

14.2.3 测试过程改进注意事项

1. 测试过程改进应结合公司实际

提到测试过程改进，很多公司一般都是设想要做得非常全面，然后投入大量的人力和物力，忽略了公司的实际情况，往往是得不偿失，结果可想而知。有句话叫“过犹不及”，做事情要把握好度。不是所有的公司都像微软公司那样对质量控制和管理能大手笔的投入，也不是所有的项目都需要像 Delphi 第一版发布时那样投入 3 万开发工程师去测试。

所以，测试过程改进时，一定要考虑测试部门的规模、公司的商业机会、企业经济实力等方面的因素，更不应该盲目跟风。合适自己的才是最好的。过程改进，一定要结合公司实际情况来进行。

2. 测试过程改进并不意味着必须投入大笔资金

测试过程改进不意味着需要大笔资金投入。有些人说我们公司不给钱买测试工具，不给任何资金请咨询公司，所以我们的测试过程改进无法取得任何进展。其实，很多情况下，工具只不过是一个辅助手段，并不足以影响到工作部署与过程改进的效果。请咨询公司固然能很快获得一些知识，但没有它们，也并不意味着过程改进就寸步难行。充分利用现有资源，发挥测试人员的智慧，一样可以不断地改进测试过程，提高测试效率。

3. 测试过程改进最好由专人负责

软件测试过程改进需要测试人员不断地跟踪测试情况、收集相关信息、确定改进内容、确定改进方法等。如果负责测试过程改进的人员身兼数职，还有很多其他的工作需要做，很容易分散精力，导致考虑不周全等。建议由专人来负责测试过程的改进比较好，测试过程改进成功的几率会大很多。

4. 测试过程改进不能急于求成

软件测试过程改进是一个循序渐进的过程，且过程改进的效果通常需要一定的时间和数据来证明。所以软件测试过程改进不能急于求成，应该正确处理好测试和测试过程改进的主次关系，一步一步用实践验证来完善。相信只要经过测试部门的不断努力，测试过程改进一定会取得成功。

14.3 测试过程改进模型

14.3.1 IDEAL 模型

为了提升企业的过程能力，SEI(Software Engineering Institute，卡内基·梅隆大学软件工程研究所)与 HP 公司合作，提出了实施 SW-CMM(Software Capability Maturity Model，软件过程能力成熟度模型)的 IDEAL 模型。IDEAL 模型是一个组织用于启动、规划和实现过程改善措施蓝图的模型，它概括了建立一个成功的过程改善项目的必要步骤，其中 I 代表 Initiating(启动阶段)，D 代表 Diagnosing(诊断阶段)，E 代表 Establishing(建立阶段)，A 代表 Acting(实施阶段)，L 代表 learning(学习阶段)(如图 14-1 所示)。

1. 启动阶段

IDEAL 模型的启动阶段是一个起点。在此阶段，初始的改进组织结构得到建立，组织结构的角色和权限进行了初步定义，初始的资源被分配。在启动阶段，初始的支持和推动组织结构将被确定。

2. 诊断阶段

组织进入软件过程的持续改进，根据组织的状况、战略业务计划、过程改进工作经验和远期目标，过程改进计划开始启动。执行评价活动以确定组织当前状态的基线。

3. 建立阶段

在改进活动中组织将待解决的问题排序，并制定解决问题的策略，根据组织的远景规

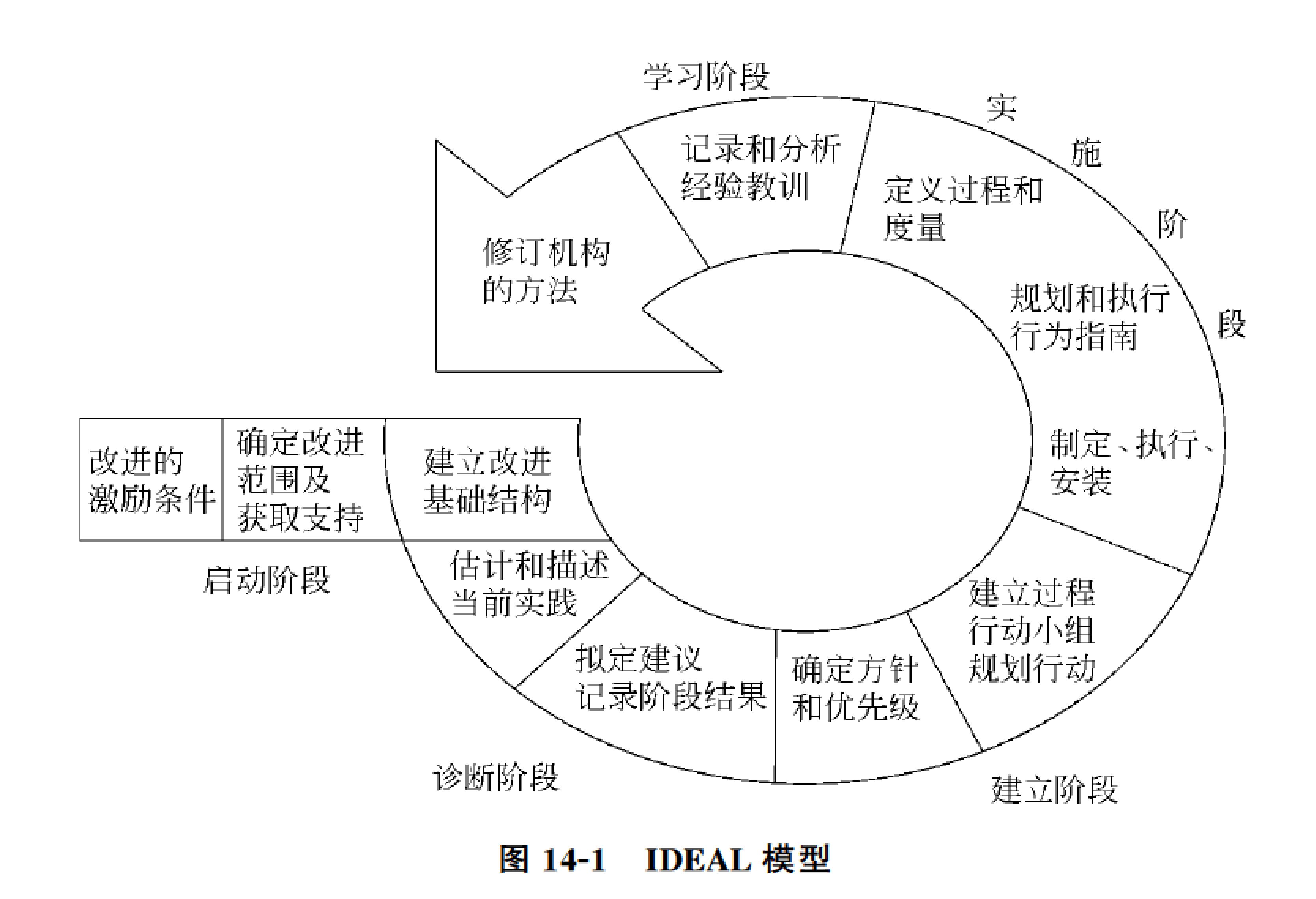


图 14-1　IDEAL 模型

划、战略商业计划、过去改进工作的经验教训、组织的业务问题和远期目标，完成过程改进行动计划草案。

4．实施阶段

在诊断阶段发现的改进方面的问题的解决方案被创建，在组织内进行试点。制定计划以执行试点去测试和评价新的或者已改进的过程。在成功试点新的过程，确认在组织级采用、开展并制度化过程所作的准备就绪之后，制定此过程在组织级进一步展开的计划并执行它。

5．学习阶段

到此时，解决方案被制定，经验教训被记录，性能和达到的目标的度量被收集，所有产品被放到过程数据库作为下一循环的信息来源。

IDEAL 是过程改进的通用模型，在实际工作中要把它作为参考，制定适合企业自身的改进模式。

14.3.2　6-Sigma 模型

6-Sigma（6σ，六西格玛）是在 20 世纪 90 年代中期开始被 GE 公司从一种全面质量管理方法演变成为一个高度有效的企业流程设计、改善和优化的技术，并提供了一系列同等的适用于设计、生产和服务的新产品开发工具。继而与 GE 公司的全球化、服务化、电子商务等战略齐头并进，成为全世界追求管理卓越性的企业最为重要的战略举措。六西格玛逐步发展成为以顾客为主体来确定企业战略目标和产品开发设计的标尺，追求持续进步的一种管理哲学。

6σ 管理法是一种统计评估法，核心是追求零缺陷生产，防范产品责任风险，降低成本，提高生产率和市场占有率，提高顾客满意度和忠诚度。6σ 管理既着眼于产品、服务质

量,又关注过程的改进。σ 是希腊文的一个字母,在统计学上用来表示标准偏差值,用以描述总体中的个体离均值的偏离程度,测量出的 σ 表征着诸如单位缺陷、百万缺陷或错误的概率性,σ 值越大,缺陷或错误就越少。6σ 是一个目标,这个质量水平意味的是所有的过程和结果中,99.999 66% 是无缺陷的,也就是说,做 100 万件事情,其中只有 3.4 件是有缺陷的,这几乎趋近到人类能够达到的最为完美的境界。6σ 管理关注过程,特别是企业为市场和顾客提供价值的核心过程。因为过程能力用 σ 来度量后,σ 越大,过程的波动越小,过程以最低的成本损失、最短的时间周期、满足顾客要求的能力就越强。6σ 理论认为,大多数企业在 3σ～4σ 间运转,也就是说每百万次操作失误在 6210～66 800 之间,这些缺陷要求经营者以销售额在 15%～30% 的资金进行事后的弥补或修正,而如果做到 6σ,事后弥补的资金将降低到约为销售额的 5%。

为了达到 6σ,首先要制定标准,在管理中随时跟踪考核操作与标准的偏差,不断改进,最终达到 6σ。现已形成一套使每个环节不断改进的简单的流程模式:界定、测量、分析、改进、控制。

- 界定:确定需要改进的目标及其进度,企业高层领导就是确定企业的策略目标,中层营运目标可能是提高制造部门的生产量,项目层的目标可能是减少次品和提高效率。界定前,需要辨析并绘制出流程。
- 测量:以灵活有效的衡量标准测量和权衡现存的系统与数据,了解现有质量水平。
- 分析:利用统计学工具对整个系统进行分析,找到影响质量的少数几个关键因素。
- 改进:运用项目管理和其他管理工具,针对关键因素确立最佳改进方案。
- 控制:监控新的系统流程,采取措施以维持改进的结果,以期整个流程充分发挥功效。

1. 西格码质量管理方法的流程

六西格码模式是一种自上而下的革新方法,它由企业最高管理者领导并驱动,由最高管理层提出改进或革新目标(这个目标与企业发展战略和远景密切相关)、资源和时间框架。推行六西格码模式可以采用由“定义、度量、分析、改进、控制”(Define,Measure,Analyze,Improve,Control,DMAIC)构成的改进流程。

DMAIC 流程可用于以下三种基本改进计划:

① 六西格玛产品与服务实现过程改进。

② 六西格玛业务流程改进。

③ 六西格玛产品设计过程改进。

这种革新方法强调定量方法和工具的运用,强调对顾客需求满意的详尽定义与量化表述,每一阶段都有明确的目标并由相应的工具或方法辅助。

推行六西格玛模式要求企业从上至下都必须改变“我一直都这样做,而且做得很好”的惯性思维。也许你确实已经做得很好,但是距六西格码模式的目标却差得很远。六西格码模式不仅专注于不断提高,更注重目标,即企业的底线收益。假设某一大企业有

1000 个基层单元，每一基层单元用六西格码模式每天节约 100 美元，一年以 300 天计，企业一年将节约 3 千万美元。通过实施模式，企业还可清晰地知道自身的水平、改进提高的额度与目标的距离等。

典型的六西格码管理模式解决方案以 DMAIC 流程为核心，它涵盖了六西格玛管理的策划、组织、人力资源准备与培训、实施过程与评价、相关技术方法的应用、管理信息系统的开发与使用等方面。

六西格玛管理战略是企业获得竞争优势和经营成功的金钥匙，在已经实施六西格玛管理并获得成功的企业名单上，你可以发现摩托罗拉、联信、美国快递、杜邦、福特这样的"世界巨人"。今天，越来越多的企业加入了"六西格玛实践者"的行列，也许这其中就有你我现在的或将来的竞争对手。

2. 西格玛水平

6σ＝3.4 失误/百万机会——意味着卓越的管理，强大的竞争力和忠诚的客户。

5σ＝230 失误/百万机会——优秀的管理、很强的竞争力和比较忠诚的客户。

4σ＝6 210 失误/百万机会——意味着较好的管理和运营能力，满意的客户。

3σ＝66 800 失误/百万机会——意味着平平常常的管理，缺乏竞争力。

2σ＝308 000 失误/百万机会——意味着企业资源每天都有三分之一的浪费。

1σ＝690 000 失误/百万机会——每天有三分之二的事情做错的企业无法生存。

6-Sigma 管理的核心特征：顾客与组织的双赢以及经营风险的降低。

六西格玛管理作为一种全新的管理模式，充分体现着量化科学管理的思想理念。六西格玛是企业走向精细化科学管理的一个质量目标，这个质量目标是企业内各个部门共同努力才能够整体实现的。

14.3.3 PDCA 模型

PDCA 循环的概念最早是由美国质量管理专家戴明提出来的，所以又称为"戴明环"。PDCA 四个英文字母及其在 PDCA 循环中所代表的含义如下：

- P(Plan)表示计划，确定方针和目标，确定活动计划；
- D(Do)表示执行，实地去做，实现计划中的内容；
- C(Check)表示检查，总结执行计划的结果，注意效果，找出问题；
- A(Action)表示行动，对总结检查的结果进行处理，成功的经验加以肯定并适当推广、标准化；失败的教训加以总结，以免重现，未解决的问题放到下一个 PDCA 循环。

PDCA 循环实际上是有效进行任何一项工作的合乎逻辑的工作程序。在质量管理中，PDCA 循环得到了广泛的应用，并取得了很好的效果，因此有人称 PDCA 循环是质量管理的基本方法。之所以将其称之为 PDCA 循环，是因为这四个过程不是运行一次就完结，而是要周而复始地进行。一个循环完了，解决了一部分的问题，可能还有其他问题尚未解决，或者又出现了新的问题，再进行下一次循环，其基本模型如图 14-2 所示。

PDCA 循环有以下四个明显特点：

1. 周而复始

PDCA 循环的四个过程不是运行一次就完结,而是周而复始地进行。一个循环结束了,解决了一部分问题,可能还有问题没有解决,或者又出现了新的问题,再进行下一个 PDCA 循环,依此类推。

2. 大环带小环

类似行星轮系,一个公司或组织的整体运行体系与其内部各子体系的关系,是大环带动小环的有机逻辑组合体(如图 14-3 所示)。

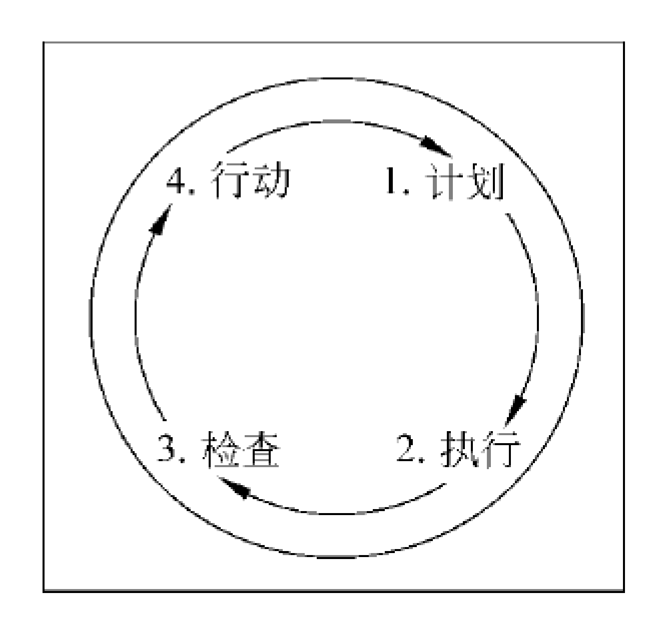


图 14-2 PDCA 模型(四个阶段)

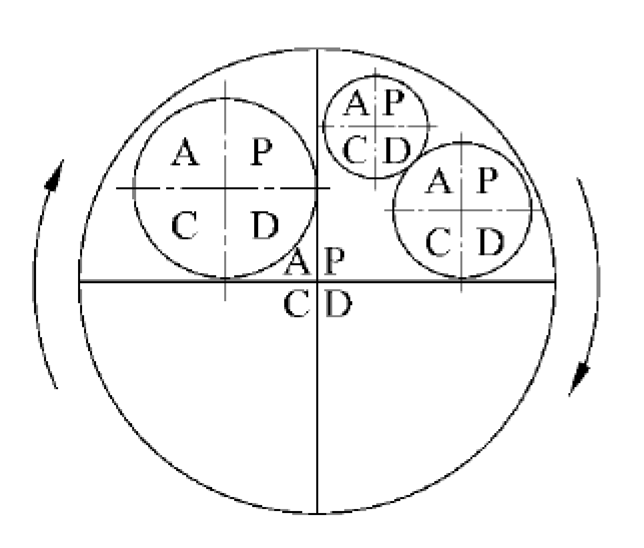


图 14-3 PDCA 模型(大环套小环)

3. 阶梯式上升

PDCA 循环不是停留在一个水平上的循环,不断解决问题的过程就是水平逐步上升的过程(如图 14-4 所示)。

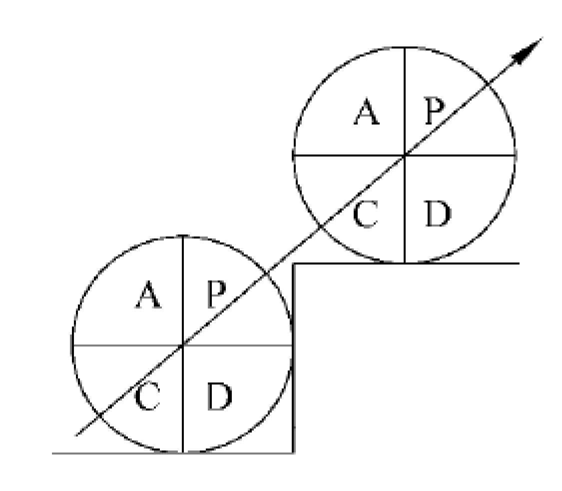


图 14-4 PDCA 模型(爬楼梯)

4. 统计的工具

PDCA 循环应用了科学的统计观念和处理方法。作为推动工作、发现问题和解决问题的有效工具,典型的模式被称为“四个阶段”、“八个步骤”和“七种工具”。

四个阶段就是 P、D、C、A。

八个步骤如下:

① 分析现状,发现问题。

② 分析质量问题中各种影响因素。

③ 分析影响质量问题的主要原因。

④ 针对主要原因,采取解决的措施。

- 为什么要制定这个措施?
- 达到什么目标?
- 在何处执行?
- 由谁负责完成?
- 什么时间完成?
- 怎样执行?

⑤ 执行，按措施计划的要求去做；

⑥ 检查，把执行结果与要求达到的目标进行对比；

⑦ 标准化，把成功的经验总结出来，制定相应的标准；

⑧ 把没有解决或新出现的问题转入下一个 PDCA 循环中去解决。

如图 14-5 所示，一个 PDCA 循环一般都要经历以下 8 个步骤。

七种工具是指在质量管理中广泛应用的直方图、控制图、因果图、散列图、趋势图、分层法和统计分析表等。

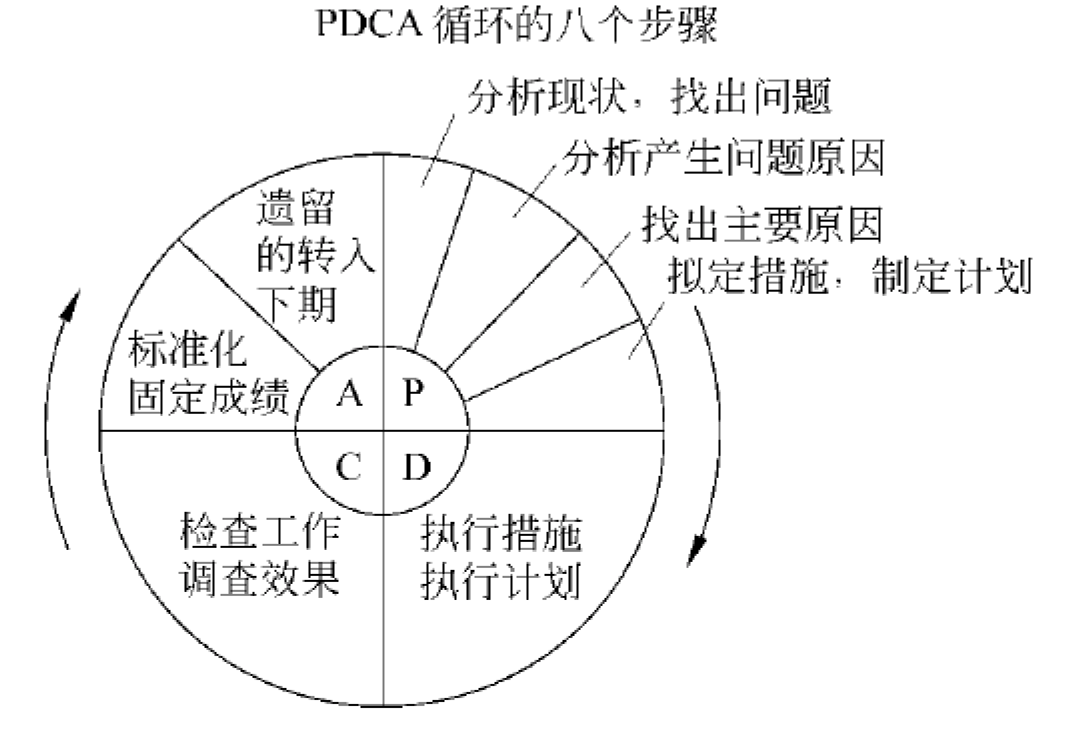


图 14-5　PDCA 模型的八个步骤

14.3.4　TMM 模型

TMM(Test Maturity Model，测试成熟度模型)是由 Burnstein 博士提出，是对 SEI-CMM 的有效补充。TMM 依据 CMM 的框架提出测试的 5 个不同级别，关注于测试的成熟度模型。TMM 描述了测试过程，是项目测试部分得到良好计划和控制的基础。

1. 第一级 初始级

TMM 初始级软件测试过程的特点是测试过程无序，有时甚至是混乱的，几乎没有妥善定义的。初始级中软件的测试与调试常常被混为一谈，软件开发过程中缺乏测试资源，工具以及训练有素的测试人员。初始级的软件测试过程没有定义成熟度目标。

2. 第二级 定义级

TMM 的定义级中，测试已具备基本的测试技术和方法，软件的测试与调试已经明确地被区分开。这时，测试被定义为软件生命周期中的一个阶段，它紧随在编码阶段之后。但在定义级中，测试计划往往在编码之后才得以制订，这显然有悖于软件工程的要求。

TMM 的定义级中需实现三个成熟度目标：制订测试与调试目标，启动测试计划过程，制度化基本的测试技术和方法。

3. 第三级 集成级

在集成级，测试不仅仅是跟随在编码阶段之后的一个阶段，它已被扩展成与软件生命周期融为一体的一组已定义的活动。测试活动遵循软件生命周期的 V 字模型。测试人员在需求分析阶段便开始着手制订测试计划，并根据用户或客户需求建立测试目标，同时设计测试用例并制订测试通过准则。在集成级上，应成立软件测试组织，提供测试技术培训，关键的测试活动应有相应的测试工具予以支持。在该测试成熟度等级上，没有正式的评审程序，没有建立质量过程和产品属性的测试度量。

集成级要实现四个成熟度目标，它们分别是：建立软件测试组织，制订技术培训计划，软件全生命周期测试，控制和监视测试过程。

在 TMM 的定义级，测试过程中引入计划能力，在 TMM 的集成级，测试过程引入控制和监视活动。两者均为测试过程提供了可见性，为测试过程持续进行提供保证。

4. 第四级 管理和测量级

在管理和测量级，测试活动除测试被测程序外，还包括软件生命周期中各个阶段的评审、审查和追查，使测试活动涵盖了软件验证和软件确认活动。根据管理和测量级的要求，软件工作产品以及与测试相关的工作产品，如测试计划，测试设计和测试步骤都要经过评审。因为测试是一个可以量化并度量的过程。为了测量测试过程，测试人员应建立测试数据库。收集和记录各软件工程项目中使用的测试用例，记录缺陷并按缺陷的严重程度划分等级。此外，所建立的测试规程应能够支持软件组中对测试过程的控制和测量。

管理和测量级有三个要实现的成熟度目标：建立组织范围内的评审程序、建立测试过程的测量程序和软件质量评价。

5. 第五级 优化，预防缺陷和质量控制级

由于本级的测试过程是可重复，已定义，已管理和已测量的，因此软件组织能够优化调整和持续改进测试过程。测试过程的管理为持续改进产品质量和过程质量提供指导，并提供必要的基础设施。

优化，预防缺陷和质量控制级有三个要实现的成熟度目标：应用过程数据预防缺陷、质量控制和测试过程优化。

14.3.5 TPI 模型

TPI(Test Process Improvement)是荣获欧洲测试杰出奖的软件测试大师 Martin Pol 和 Tim Koomen 提出的软件测试过程改进模型，TPI 模型目前是西方工业界和软件公司、研究机构普遍采用的测试过程方法，并基于 TPI 建立了其测试体系和测试规范；TPI 模型能够检视组织的测试过程成熟度，帮助定义渐进的和可控的改进步骤，应用 TPI 模型，能够帮助企业解决许多问题，例如：如何加快产品上市时间；如何优化测试资源，降低测试成本；如何定义可控可管理的测试流程；如何更好更快地检视产品质量；如何减少对核心测试人员的依赖；如何实现测试过程中充足的自动化等。

通过 TPI 模型，能够确定测试过程的当前状态，下一个需要进行过程改进的地方，以及推荐的改进步骤。TPI 模型大部分都使用了类似于 CMM 的语言，大多数的术语和 CMM 中的类似，这使得当前 CMM 的支持者使用 TPI 模型时比较容易。TPI 模型主要部分如图 14-6 所示，包括关键域、等级、检查点和改进建议。

1. 关键域

在 TPI 模型中，测试过程分为 20 个测试组织需要明确的关键域，改进基线和改进建议是基于以下 20 个关键区域的。分别是测试策略、生命周期模型、引入时间、估计和计划、测试规格技术、静态测试技术、度量、测试自动化、测试环境、办公环境、承诺与动力、测试功能和培训、方法的范围、交流、报告、缺陷管理、测试件管理、测试过程管理、评价和低

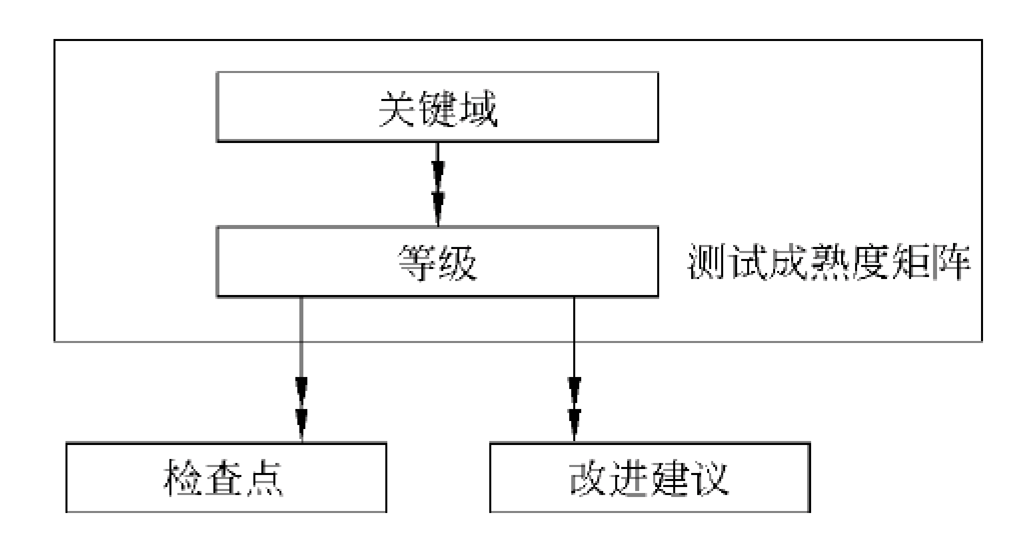


图 14-6　TPI 组成

级测试。

2. 成熟度等级

对测试过程改进建立基线可以基于整个测试过程，或者若干个关键域，通过对每一个关键域的检查可以将测试过程划分为一个成熟度级别，测试过程成熟度级别共分为 14 个级别，从 0 到 13 成熟度逐级增加。而每一个关键域分为 1～4 个成熟度级别，这些级别分别用 A、B、C、D 来标明，从 A 到 D 表示成熟度增加。

3. 测试成熟度矩阵

对于测试过程基线的评估是基于测试成熟度矩阵模板，TPI 评估的空白表格如下表。

表 14-1　TPI 评估成熟度矩阵

关键域	0	1	2	3	4	5	6	7	8	9	10	11	12	13
测试策略		A					B				C		D	
生命周期模型		A			B									
引入时间			A				B				C		D	
估计和计划				A							B			
测试规格技术		A		B										
静态测试技术					A		B							
度量						A			B			C		D
测试自动化				A				B			C			
测试环境				A				B						C
办公环境				A										
承诺与动力		A				B						C		
测试功能和培训				A			B			C				
方法的范围					A						B			C
交流			A		B							C		

续表

关键域	0	1	2	3	4	5	6	7	8	9	10	11	12	13
报告		A			B		C					D		
缺陷管理		A				B		C						
测试件管理			A			B				C				
测试过程管理		A		B								C		
评价							A			B				
低级测试					A		B		C					

测试成熟度被分为 3 个等级：

- 1～5 为受控的；
- 6～10 为有效的；
- 11～13 为优化的。

当组织达到某一过程域的某一级别，将该部分的表格用阴影覆盖，通过比较阴影最多的部分，可以将组织划分为受控的、有效的、优化的，这样有利于组织对测试成熟度等级有直观的认识。

在测试成熟度矩阵中，所有的关键域和等级对整个测试过程的完成并不是同等重要的，而不同的关键域和等级之间存在着相关性。因此，所有的关键域和等级在测试成熟度矩阵中是相互联系的。等级在模型中并没有垂直对齐，这是因为一个关键域的起始等级可能原来就在另一个关键域的同一级别前面。比如说，缺陷管理的 A 级在度量的 A 级前面是因为，要使用缺陷度量的话，就需要收集缺陷信息。表 14-2 是各关键域与等级的相关性表，括号中的数字表示所依赖的关键域的序号，字母表示该关键域的等级。

表 14-2　评估标准

关键域	A 级	B 级	C 级	D 级
测试策略	单个高级测试的策略(5A,11A)	高级测试的组合策略（2A，5B，11B，14B,18B)	高级测试和低级测试/评价的组合策略(20C 或(3C，19B))	所有等级的测试和评价的组合策略（3C，19B,20C)
生命周期模型	计划，规格说明，执行(11A)	计划，准备，规格说明，执行和完成(6A，17A)		
引入时间	测试基础的完成(2A)	测试基础的开始(2B)	需求定义的开始	项目初始化
估计和计划	实质性的估计和计划(2A)	经统计证实的估计和计划(7B,15B)		
测试规格技术	非正式技术	正式技术（12A，17A)		

续表

关键域	A 级	B 级	C 级	D 级
静态测试技术	基础测试审查	检查列表		
度量	项目度量(产品)(11B, 15B, 16A, 18B)	项目度量(过程)(15C,16B)	系统度量(13B, 14C,18C)	组织度量(>1个系统)
测试自动化	工具的使用	已管理的测试自动化(5A或5B,12A)	优化的测试自动化	
测试环境	已管理的和受控的测试环境(12A)	在最适合的环境中进行测试(1B)	随时可供使用的环境	
办公环境	足够的适时的办公环境			
承诺与动力	预算和时间的分配	测试集成到项目组织中(2A,15B,16A, 18B)	测试工程(1C,3C, 8B,15C)	
测试功能和培训	测试经理和测试人员	方法、技术和功能支持、管理	正式的内部质量保证(13A)	
方法的范围	特定于项目(2A,5B, 16A,17A,18B)	组织通用的	组织优化,研发活动(11B,18C)	
交流	内部交流	项目交流(2A,15B, 16A)	组织内关于测试过程质量的交流(13B)	
报告	缺陷	进展(测试状态和产品),活动(费用和时间,里程碑),缺陷及其优先级(2A,16A, 18B)	度量已正式的风险和建议(1A,5B, 7A,16B)	建议具有软件过程改进的特征(1C,11C)
缺陷管理	内部缺陷管理	广泛的缺陷管理,并使用灵活的报告措施	项目缺陷管理	
测试件管理	内部测试件管理	测试基础和测试对象的外部管理	可重用的测试件(5B)	对测试用例的系统需求的可追踪性
测试过程管理	计划和执行	计划、执行、监控和调整	组织内监控和调整(13B)	

续表

关键域	A 级	B 级	C 级	D 级
评价	评价技术	评价策略		
低级测试	低级测试生命周期模型(计划、规格说明和执行)	白盒测试	低级测试策略	

4．检查点

检查点是用来判定成熟度级别的问题。为了保证客观性，每一个级别设置多个检查点。如果一个关键域通过了所有检查点的同一个等级，那么这个关键域就被定为这个等级。

例如，测试自动化关键域的检查点如下：

A 级，工具的使用：使用自动化测试工具实现缺陷管理和计划、控制等活动。

B 级，已管理的测试自动化：

① 至少使用了两种自动化工具用于测试执行。

② 测试团队掌握测试工具的使用效率。

③ 已达到测试功能和培训 A 级—测试经理和测试人员。

④ 已达到测试规格技术 B 级—正式技术，只有在测试脚本易于维护时，自动化测试才会产生效果。

C 级，优化的测试自动化：

① 使用自动化测试工具实现计划、准备、规格说明和执行阶段的管理。

② 测试团队掌握测试工具的使用效率。

5．改进建议

对测试过程的改进建议是依据测试前景制定的。对于没有达到目标等级的关键域，合理地改进步骤是基于关键域的检查点的，当实现了所有的检查点，并且其所依赖的关键域达到了要求的等级，该关键域即可达到目标等级，实现了改进。

本章小结

软件测试度量是测试过程改进的依据，本章首先介绍了有关软件测试度量的有关内容，主要是测试度量的概念、度量内容、度量分类、度量过程等；测试过程改进部分介绍了软件过程改进的改进内容、改进过程、改进策略、改进注意事项，以及常见的软件过程改进模型。

软件测试过程度量是对软件测试过程的量化分析，目的是用软件度量学的方法来科学地评价软件测试质量，控制和管理软件测试过程，合理组织和分配资源，以较低的成本获得高质量的软件；软件测试度量主要包括进度度量、成本度量、测试质量度量、产品质量度量等；测试度量过程首先确定度量信息需求，然后制定针对性的度量计划，执行度量，最

后对度量进行评估。

测试过程改进内容主要包括测试管理改进、测试技术改进、测试人员改进等;测试过程改进过程首先是确定测试过程改进目标,然后制定相应的测试过程改进计划,并建立相应的跟踪控制机制,然后实施测试过程改进,最后根据改进反馈结果进行总结;测试过程改进策略和注意事项主要是结合自己的实际情况,制定适合自己的测试过程改进过程;常见的测试过程改进模型主要有 IDEAL 模型、6-Sigma 模型、PCDA 模型、TMM 模型、TPI 模型等。其中前三个模型并不是测试领域的专有模型,但是其过程改进的思想却是很值得借鉴的,后面的 TMM 和 TPI 是软件测试改进的特有模型,有着不错的应用前景。

PART 5

第五篇

软件测试工具及其应用

本篇主要介绍目前软件测试行业应用最广泛的功能测试工具、性能测试工具、测试管理工具，对其测试流程、基本使用方法、测试中的常见问题及解决方法等进行介绍，并附有项目实例。同时，简单介绍了软件测试工具的分类、选择及其工作原理，并给出了目前各类软件测试工具的简介列表。

本篇名词解释：

吞吐量（throughout）：是指单位时间内系统处理的客户请求的数量，直接体现软件系统的性能承载能力。一般来说，吞吐量用请求数/秒或页面数/秒来衡量；从业务的角度，吞吐量也可以用访问人数/天或处理的业务数/小时等单位来衡量；从网络的角度来说，也可以用字节数/天等单位来考察网络流量。

思考时间（think time）：也被称为“休眠时间”，从业务的角度来说，这个时间指的是用户在进行操作时，每个请求之间的间隔时间。

事务（transaction）：在软件测试中，事务的概念指的是客户端向服务器端发送的一项或一系列的请求，如整个下订单业务可以视为一个事务。

检查点（checkpoint）：为了检验测试脚本运行的准确性，在测试脚本中插入的检查语句，检验测试脚本运行结果与预期结果是否一致。

集合点(**rendezvous**):为了使不同数量的用户并发执行同一或不同的功能操作,在操作前插入的等待语句,使得不同的用户按照制定的集合策略同时进行接下来的操作。

关联(**correlation**):就是把脚本中某些固定的数据,转变成服务器端返回的、动态的、每次都不一样的数据,如客户端与服务器端会话的SessionID。

测试场景(**test scenario**):主要是模拟软件系统一些实际的应用情况,包括测试时执行的业务,每种业务执行的用户数量,模拟的总用户数、数据库容量,用户增长方式、测试循环方式、用户退出方式、执行过程中的相关参数设定,如思考时间等,还应有性能指标,如资源利用率、响应速度、吞吐量等。

CHAPTER

第 15 章 软件测试工具及其分类

随着人们对软件质量的重视程度的提高,软件测试的地位逐步提高,软件测试技术也随之快速发展,逐渐从过去手工作坊式的测试向测试自动化的方向发展,引发了软件测试工具的研究和应用热潮。在测试过程中引入测试工具,至少带来以下好处。

1. 提高工作效率

这是引入测试工具的一个显著好处。一些固定的、重复性的测试工作,可以由测试工具来完成,这样就使得测试人员能有更多的时间来计划测试过程,设计测试用例,使测试工作更加完善。

2. 保证测试的准确性

测试是需要投入大量的时间和精力的,人工进行测试时,经常会出现一些人为的错误,而工具的特点恰恰能保证测试的准确性,防止人为疏忽造成的错误。

3. 执行手工很难完成的测试工作

有一些测试工作,通过手工难于执行,或者是无法执行,有的是因为进行起来较为复杂,有的是因为测试环境难以实现。而测试工具则可以满足对特殊测试要求的测试工作,如模拟大量用户执行并发操作等。

15.1 软件测试工具分类

现在的软件测试工具很多,基本上覆盖了各个测试阶段,贯穿于整个软件开发生命周期,覆盖了分析设计、需求管理、配置管理、测试管理、缺陷管理、功能测试、性能测试、单元测试等方面。

15.1.1 按照原理分类

根据测试工具原理的不同,软件测试工具可分为静态测试工具、动态测试工具和其他支持测试活动的工具,每类测试工具在功能和其他特征方面具有相似之处,支持一个或多

个测试活动(表 15-1 为测试工具分类表。)。

表 15-1　软件测试工具分类表

工具类型	功能和特征说明	举例	备注
静态测试工具	对软件需求、结构设计、详细设计和代码进行评审、走查和审查的工具	复杂度分析、数据流分析、控制流分析、接口分析、句法和语义分析等工具	针对软件需求、结构设计、详细设计的静态分析工具很少
动态测试工具	支持执行测试用例和评价测试结果的工具,包括支持选择测试用例、设置环境、运行所选择测试、记录执行活动、故障分析和测试工作有效性评价等	覆盖分析、捕获和回放、存储器测试、变异测试、仿真器及性能分析、测试用例管理等工具	测试捕获和回放及数据生成器可用于测试设计
测试支持工具	支持测试计划、测试设计和整个测试过程的工具	测试计划生成、测试进度和人员安排评估、基于需求的测试设计、测试数据生成、问题管理和测试配置管理等工具	复杂度分析可用于测试计划的制定,捕获和回放、覆盖分析可用于测试设计与实现

1. 静态测试工具

静态测试工具采用静态测试方法,主要通过对软件源代码进行分析,找出软件所存在的结构设计和代码质量方面的缺陷。静态测试工具对软件源代码进行语法扫描,找出不符合编码规范的地方,并可以根据某种质量模型评价代码的质量,另外还可以通过源码分析自动生成系统的调用关系图、逻辑控制图等。静态测试工具不需要运行软件,所以也就不需要对代码编译链接,生成可执行文件。

典型的静态测试工具有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。

2. 动态测试工具

动态测试工具采用动态测试方法对软件进行测试,通过向代码编译生成的可执行文件中插入一些监测代码的方式,用来统计程序运行时的数据。其与静态测试工具最大的不同就是动态测试工具要求被测系统实际运行。动态测试工具又可以分为白盒测试工具和黑盒测试工具。比如,IBM 的 Rational Test RealTime 就是一款典型的动态测试工具。

1) 白盒测试工具

白盒测试工具一般是针对代码进行测试,测试中发现的缺陷可以定位到代码级。白盒测试也称结构测试或逻辑驱动测试,它是按照程序内部的结构测试程序,通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行,检验程序中的每条通路是否都能按预定要求正确工作。

白盒测试是把测试对象看做一个打开的盒子,测试人员依据程序内部逻辑结构相关信息,设计或选择测试用例,对程序所有逻辑路径进行测试,通过在不同点检查程序的状态,确定实际的状态是否与预期的状态一致。

典型的白盒测试工具有开源的 xUnit 系列测试工具、ParaSoft 系列测试工具、Compuware 系列测试工具等。

2）黑盒测试工具

黑盒测试把软件看做一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在程序接口进行测试，只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构，不考虑内部逻辑结构。黑盒测试工具包括功能测试工具和性能测试工具。黑盒测试工具可以大大减轻测试的工作量，可以应用于回归测试。

典型的黑盒测试工具的代表有 HP 公司的 LoadRunner、WinRunner、QuickTest Professional，IBM 公司的 Robot，Compuware 公司的 QACenter，Radview 公司的 WebLoad、Microsoft 公司的 WebStress 等工具。

3）测试支持工具

测试支持工具不用于执行测试，而是对测试提供相应的支持。测试支持工具主要指的是测试管理工具，如配置管理、缺陷管理、测试过程管理等；还包括一些其他测试辅助工具，如为测试执行提供数据支持的数据生成工具（如 TestBytes）等。

15.1.2　按照用途分类

前面我们从原理的角度对测试工具做了一个分类，另外根据工具所完成的任务不同，测试工具还可以分为测试设计工具、单元测试工具、功能测试工具、性能测试工具、测试过程管理工具等。

1. 测试设计工具

测试设计工具，主要是用于测试用例设计的工具。在测试设计时，很多设计测试用例的原则、方法是固定的，比如等价类划分、边界值分析、因果图等，这些成型的方法，很适合通过软件工具来实现。

测试用例设计工具按照生成测试用例时数据输入内容的不同，可以分为基于程序代码的测试用例设计工具和基于需求说明的测试用例设计工具。

1）基于程序代码的测试用例设计工具

基于程序代码的测试用例设计工具是一种白盒测试工具，它读入程序代码文件，通过分析代码的内部结构，产生测试的输入数据。这种工具一般应用在单元测试中，针对的是函数、类这样的测试对象。由于这种工具与代码的联系很紧密，所以，一种工具只能针对某一种（些）编程语言。

这类工具只能产生测试的输入数据，而不能产生输入数据后的预期结果，这个局限也是由这类工具生成测试用例的机理所决定的。所以，基于程序代码的测试用例设计工具所生成的测试用例，还不能称之为真正意义上的测试用例。即使这样，这种工具仍然为设计单元测试的测试用例提供了很大便利。

2）基于需求说明的测试用例设计工具

基于需求说明的测试用例设计工具依据软件的需求说明，生成基于功能需求的测试用例。这种工具所生成的测试用例既包括了测试输入数据，也包括预期结果，是真正完整的测试用例。

使用这种测试用例设计工具生成测试用例时,需要事先人工地将软件的功能需求转化为工具可以理解的文件格式,再以这个文件作为输入,通过工具生成测试用例。在使用这种测试用例设计工具来生成测试用例时,需求说明的质量是很重要的。

由于这种测试用例设计工具是基于功能需求的,所以可用来设计任何语言、任何平台的任何应用系统的测试用例。

2. 单元测试工具

单元测试是软件测试过程中一个重要的测试阶段。与集成测试、验收测试相比,在编码完成后对程序进行有效的单元测试,能更直接、更有效地改善代码质量。

进行单元测试时,根据被测单元(这个单元可能是一个函数,或者是一个类,甚至是一个类簇)的规格说明,设计测试用例,然后通过执行测试用例,验证被测单元的功能是否正常实现。除此之外,在单元测试阶段,我们还需要找出那些短时间不会马上表现出来的问题(如 C++ 代码中的内存泄露),还需要查找代码中的性能瓶颈,并且为了验证单元测试的全面性,我们还想了解单元测试结束后,我们的测试所达到的覆盖率。

典型的单元测试工具有以下几类。

1) 动态错误检测工具

用来检查代码中类似于内存泄露、数组访问越界这样的程序错误。

2) 性能分析工具

用来记录被测程序(小到一行代码、一个函数的运行时间,大到一个 exe 或 dll 文件)的执行时间,帮助定位代码中的性能瓶颈。

3) 覆盖率统计工具

统计出我们当前执行的测试用例对代码的覆盖率。覆盖率统计工具提供的信息,可以帮助我们根据代码的覆盖情况,进一步完善测试用例,使所有的代码都被测试到,保证单元测试的全面性。

常见的单元测试工具有 Compuware 公司的 NuMega DevPartner Studio、IBM 公司的 Rational Suite Enterprise 等。另外,在开源社区也有一些久负盛名的单元测试框架,如最有影响力的 xUnit 系列,已经演化延伸到多个编程语言,如 Java 语言的 JUnit、C++ 语言的 CppUnit 等。

3. 功能测试工具

功能测试自动化工具理论上可以应用在各个测试阶段,但大多数情况下是在验收测试阶段中使用。功能测试自动化工具的测试对象主要是那些拥有图形用户界面的应用程序。

功能测试工具主要是用来进行重复的、复杂的功能测试工作,如重复的数据输入、复杂的工作流程等。功能测试工具一般都支持测试脚本的复用,可以提高测试的效率,特别是在回归测试中,更能发挥功能测试工具的优势。

功能测试工具主要是将软件的操作过程记录下来,生成测试脚本,然后支持对测试脚本的增强,包括加入检查点、将固定值参数化等,以达到各种操作的要求,然后回放准备的测试脚本,模拟软件的各种操作过程,记录相应的结果,并将测试结果与理想的结果进行

比较,确定运行是否成功。

功能测试自动化工具是软件测试工具中非常活跃的一类工具,现在已经较为成熟,市面上也有一些商业软件供选择,如 HP-Mercury 公司的 WinRunner、QuickTest Professional,IBM 公司的 Robot 等,都是广受好评的功能测试自动化工具。

4. 性能测试工具

性能测试工具,主要用于软件的性能考察,通过性能测试检验软件的性能是否达到预期要求,是软件产品测试过程中的一项重要任务。性能测试用来衡量系统的响应时间、事务处理速度和其他时间敏感的需求,并能测试出与性能相关的工作负载和硬件配置条件。

性能测试工具还可以应用于软件性能故障定位。软件系统通过实际应用,或者通过性能测试工具测试,发现系统可能存在的性能问题,如响应时间过长、内存泄露等。性能测试工具通过自动化测试手段,可以反复测试,复现故障,同时提供详细的测试数据,供测试人员进行综合分析,从而确定出软件的故障原因。

性能测试工具还可以用于性能调优。软件出现性能故障后,经过分析,确定出故障原因,需要进行不断地调整、验证,性能测试工具可以很好地满足这种要求,以确保软件在最优状态下运行。

同功能测试工具一样,性能测试工具也是通过测试脚本来记录软件的操作过程,性能测试工具一般包括测试脚本生成器、测试控制器、负载生成器、结果分析器等几部分。

1) 脚本生成器

用来协助录制开发测试脚本,实现对测试脚本的增强,如将数据参数化、定义事务、加入集合点、加入检查点、进行关联等,来达到模拟真实应用环境下的操作的要求。

2) 测试控制器和负载生成器

测试控制器和负载生成器用来模拟各种测试场景。负载生成器可以生成多个虚拟用户,这些虚拟用户代替人工操作去执行测试脚本,测试控制器可以控制一台或多台负载生成器,并且可以监控测试场景的执行过程,设置加压策略、并发策略等,同时监控处理场景运行期间的异常错误、虚拟用户运行日志等。

3) 结果分析器

结果分析器负责将性能测试工具收集的原始测试数据自动分析整理,统计计算出各种性能指标值,如平均响应时间、吞吐量、服务器 CPU 平均利用率等,并给出一些测试数据的各类统计图表,供测试人员进行分析。

性能测试自动化工具是软件测试工具中备受关注的一类工具,典型的有 HP-Mercury 公司的 LoadRunner、IBM 公司的 Robot(既可用于功能测试,也可用于性能测试)等、Compuware 公司的 QALoad、RedView 公司的 WebLoad(WebLoad 目前已经开源)等。

5. 测试管理工具

测试管理工具主要用于整个测试过程的控制和管理。软件测试贯穿于整个软件开发过程,在每个阶段都有一些数据需要保存,人员之间也需要进行交互。测试过程管理工具就是一种用于满足上述需求的软件工具,它管理整个测试过程,在不同测试阶段产生的文

档、数据都可以在系统中得到保存和控制，使各种角色借助于集成管理系统可以协同的工作。

测试过程管理工具一般具有管理软件测试需求、管理测试计划、管理测试用例、缺陷跟踪、测试过程中各类数据的统计和汇总等功能。常见的测试管理工具有 HP 公司的 Test Director、Quality Center，开源的 Subversion（SVN）、Bugzilla 等。

15.2 软件测试工具的实现原理

软件测试工具实质是在模拟测试人员的实际测试过程，模拟测试人员对计算机的操作过程和行为，包括数据请求和接收，或者类似于编译系统那样对计算机程序进行静态检查。

1. 静态测试工具

静态测试工具一般是对代码进行语法扫描，在工具中描述类、对象、函数、变量、运算等定义规则、语法规则等，在分析时，对代码进行语法扫描，找出不符合编码规范的地方，再根据某种质量模型评价代码的质量，生成系统的调用关系图等。

2. 动态测试工具

1）白盒测试工具

白盒测试工具一般采用类似于高级编译系统的代码分析方式进行测试，一般是针对不同的高级语言去构造相应的分析工具。测试工具直接访问被测试的应用程序的代码，对其中的类和函数进行调用，输入各种测试数据，检查函数的返回值，通过比较返回值与期待的值是否一致来判断测试是否通过。

为了更好地进行代码分析，可以在代码中设置一些“断点”，在这些断点和其他地方插入一些检测代码，存于构造的可执行文件中，随时了解这些关键点、关键时刻的某个变量的值、内存、堆栈状态等。

2）黑盒测试工具

黑盒测试工具一般将捕获的用户操作过程生成测试脚本，并提供相应功能对测试脚本进行编辑，达到各种测试的要求，并提供自动比较功能，来比较测试结果与理想结果是否一致，从而实现对软件的自动化测试。

（1）捕获用户操作。首先测试工具将用户的每一步操作都记录下来，如功能测试工具记录被操作的程序的显示对象，如窗口、按钮等，或者实际的物理位置，以及相对应的操作、状态变化、属性变化等；性能测试工具则是把客户端的请求内容记录下来。将所有的记录转化为测试脚本，用于描述每一步的操作过程。

（2）脚本技术。脚本作为一种特殊的计算机程序，包含数据和指令。指令作为控制信息来操作软件中的对象，数据则主要是被操作对象属性的值。脚本技术就是围绕着脚本程序结构而进行的设计，测试工具提供相应的编辑功能，可以实现测试用例所要求的输入、步骤和验证点，以及同步点、固定值参数化等，来达到各种测试的要求，如不同的输入数据，不同的路径等。

同时,测试功能还提供相应的调试功能,如断点、单步运行等,支持对测试脚本进行调试,保证增强后的测试脚本可以正常运行。

回放测试脚本时,将脚本语言所描述的过程转化为屏幕上的操作,从而实现对软件的自动化测试。

(3) 结果比较。在测试过程中运行测试脚本,将捕获的测试结果与预先准备的输出结果进行比较,从而确定测试用例是否通过。自动比较可以对比分析屏幕或屏幕区域图像,比较窗口或窗口上空间的数据或属性,比较网页、文件等。

15.3 软件测试工具的选择原则

测试工具具有众多优点,可以提高工作效率、保证测试的准确性、执行手工很难完成的测试工作等。现在各种各样的软件测试工具非常多,面对如此多的测试工具,对工具的选择就成了一个比较重要的问题。

下面介绍几个案例。

案例一:某公司准备开发一套软件,由于测试功能比较多,且流程比较复杂,为了在紧张的时间内保证该软件顺利的上线使用,该公司决定引入自动化测试,购买了市场上应用广泛的一款自动化测试工具。然而,由于在整个开发过程中需求和用户界面变动较大,导致每次的测试脚本不能复用,不仅不能减轻工作量,反而加重了测试人员的负担。

案例二:某公司开发了一套应用软件,采用的开发语言是 C#,该公司有一套自动化测试工具,他们选用 WinRunner 做自动化测试。结果在测试中,经常会遇到 WinRunner 识别对象的问题,导致花费很长的时间在测试脚本调试,而测试脚本运行失败。后来发现原因是 WinRunner 不支持基于.NET 的应用软件。

案例三:某公司开发的是一种 C/S 结构的应用软件,为了了解该软件的性能状况,欲对该软件进行性能测试,该公司选择了功能强大、应用广泛的性能测试工具 LoadRunner。测试人员根据测试要求,开始编写性能测试计划和性能测试用例;设计完毕后,开始着手进行测试。在准备测试脚本时,发现用 LoadRunner 根本就无法进行测试,原因是该应用软件是开发人员自己写的通信协议,而且在该应用软件中有很多插件。

对以上案例进行分析,我们可以发现,上述三个公司没有考虑到自己的实际情况,盲目引入测试工具,导致最终无法达到理想的结果。

以上都是较为常见的现象,对测试工具存在片面的认识,认为具有强大功能的测试工具一定能够解决面临的所有问题。这种想法忽视了除技术以外我们仍然需要做的工作。软件测试工具确实具有很多优点,能提高测试的效率和质量,但它并不是能够解决一切问题的灵丹妙药。利用自动化测试工具进行测试只是对手工测试的一种补充,测试工具不能完全代替手工测试。

选择好测试工具,最流行的工具不一定适合自己,真正适合自己的工具才是最好的。在考虑选用工具的时候,先理清自己到底想要什么,关心哪些问题,然后再去查看各种测试工具,主要注意以下几个方面。

1. 适用范围

软件测试工具选型的第一步就是看被测软件系统是否在该工具的使用范围之内，否则自动化实施就会功败垂成。主要是明确以下内容：

- 是否适用于被测软件系统的开发语言环境；
- 是否适用于被测软件系统的运行环境，包括软件环境、硬件环境；
- 是否支持被测系统使用的通信协议。

2. 功能

功能是选择一个测试工具重点关注的内容。测试工具提供的功能不一定是越多就越好，在实际的选择过程中，适合自己的才是根本。事实上，目前同类的软件测试工具之间的基本功能都是大同小异，各种软件提供的功能也大致相同，只不过有不同的侧重点。

(1) 白盒测试工具的选择，可以考虑以下几个方面：

- 代码覆盖的深度；
- 可视化程度，这点对工作量巨大并且枯燥的白盒测试工作很有必要，如测试过程中是否使用不同的颜色区分已执行和未执行的代码段。

(2) 黑盒测试工具的选择，可以注意以下几个方面：

- 采用什么样的测试脚本语言；
- 脚本调试功能是否强大，都有哪些具体的功能；
- 是否可以自动收集测试数据，并对测试数据进行相应的处理，如生成报表、曲线图等；
- 是否支持特殊的应用要求，如防火墙、负载均衡等；
- 是否可以对相关测试数据进行实时监控等。

3. 价格

除了功能之外，价格就应该是最重要的因素了。需要注意的是，有些工具根据提供的功能不同，价格也不同。如功能测试工具支持的插件，性能测试工具支持的协议、虚拟用户数等。

4. 与其他工具的集成

测试工具引入的目的是测试自动化，引入工具需要考虑工具引入的连续性和一致性。所以在选择测试工具时，需要考虑该测试工具与其他工具的集成能力如何，包括与开发工具的集成，以及与其他测试工具的集成。

软件测试在整个软件开发过程中占据了将近一半的时间和资源，通过在测试过程中合理地引入软件测试工具，能够缩短软件开发时间，提高测试质量，从而更快、更好地为用户提供他们需要的软件产品。

本章小结

本章主要介绍了软件测试工具的分类、软件测试工具的实现原理以及软件测试工具

的选择原则。

软件测试工具根据原理不同,可以分为静态测试工具、动态测试工具、测试支持工具。静态测试工具可以对工程源码清单直接进行分析;动态测试工具可分为白盒测试工具和黑盒测试工具;测试支持工具包括测试管理工具和一些专用的测试工具,如针对数据库测试工具等。根据工具所完成的任务,测试工具又可以分为测试设计工具、单元测试工具、功能测试工具、性能测试工具、测试过程管理工具等。

软件测试工具的实现原理实质是在模拟测试人员的实际测试过程,模拟测试人员对计算机的操作过程和行为,包括数据请求和接收,或者类似于编译系统那样对计算机程序进行静态检查。静态测试工具根据一定的规则对代码进行扫描;动态测试工具中的白盒测试工具一般采用类似于高级编译系统的代码分析方式进行测试,一般是针对不同的高级语言去构造相应的分析工具;动态测试工具中的黑盒测试工具一般将捕获的用户操作过程生成测试脚本,并提供自动比较功能,来比较测试结果与理想结果是否一致,从而实现对软件的自动化测试。进行软件测试工具的选择时,需要综合考虑如下几个方面:适用范围、测试工具功能、测试工具的价格、与其他测试工具的集成情况等。

CHAPTER

第 16 章 功能测试工具

功能测试工具是软件测试过程中常用的一类工具，它可以模拟用户对软件操作的过程，可以用来代替人的部分工作，特别是在重复的、复杂的功能测试工作中，功能测试工具的作用尤为明显。目前，功能测试工具较多，有适用范围比较广泛的测试工具，也有在某一领域专用的测试工具，有成熟的商业测试工具，也有开源的测试工具。本章我们主要选择介绍 HP 公司的 WinRunner 和 QuickTest Professional，这是两款目前应用比较广泛的商业功能测试工具。本章将从其功能特点、使用方法、常见问题等方面加以论述，希望通过本章的学习能够使读者了解功能测试自动化的一般原理，同时掌握工具的基本使用方法。

16.1 WinRunner

16.1.1 概述

WinRunner 是 HP 公司的一款企业级的功能测试工具，用于检验企业应用程序是否能顺利运行。通过自动捕获、检测和重放用户的交互操作，WinRunner 能够发现系统缺陷，确保应用程序顺利部署，并且能够维持其长时间的可靠运行。

WinRunner 具有如下功能特点：

① 可以快速完成功能点的测试。用 WinRunner 创立一个测试，只需记录下一个标准的业务流程，如下一张订单等。WinRunner 直观的记录流程可以使得测试人员在 GUI 上轻轻点击鼠标就可建立测试。WinRunner 可以针对相同测试脚本，重复执行相同的动作，从而消除人工测试所带来的理解上的误差，还可以通过直接编辑测试指令来满足各种复杂测试的需求。

② 拥有很好的测试脚本复用性。WinRunner 支持程序风格的测试脚本，通过使用通配符、宏、条件语句、循环语句等，可以较好地完成测试脚本的重用，这在软件升级后的测试中非常有效，这样可大大地节省时间和资源，充分利用测试投资。

③ 适用范围广泛。针对大多数编程语言和 Windows 技术，WinRunner 提供了较好的集成、支持环境，这对基于 Windows 平台的应用程序实施功能测试提供了极大的便利。目前有相当数量的企业应用程序仍然使用非标准的对象，WinRunner 能识别以前未知的对象，不必特别编写代码。

④ 良好的测试分析结果。WinRunner 的互动式报告工具会列出在测试中发现的错误和出错的位置，通过提供这种详尽的、易读的报告，WinRunner 可以帮助您解释所得的结果。这些报告对在测试运行中发生的重要事件进行描述，如错误内容和检查点等。单击按钮，您还能进一步获取任何未被包括在此测试范围内的错误的详尽资料。

16.1.2 WinRunner 的应用

1. 使用概述

1）WinRunner 启动

选择“开始”→“程序”→ WinRunner → WinRunner 命令，启动 WinRunner。WinRunner 启动过程中，会出现插件加载窗口，用户可以在此选择需要载入的测试支持插件（如图 16-1 所示）。





图 16-1　WinRunner 启动

根据被测试软件，选择插件类型，单击 OK 按钮，启动 WinRunner。WinRunner 的记录/执行引擎（Record/Run Engine）的图标出现在 Windows 的任务条上，该引擎是用来建立和维护 WinRunner 和被测软件之间的连接。

在 WinRunner 启动时，可以选择支持 ActiveX control、PowerBuilder、VisualBasic 或 WebTest 等的插件。其他插件需要单独再向 MI 公司购买，建议不要同时载入所有的插件，不必要的插件可能会对录制或执行脚本造成问题。

2）WinRunner 主窗口

WinRunner 的主窗口包括以下部分。

- 标题栏（Title Bar）：显示当前的测试脚本路径及名称；

- 菜单栏(Menu Bar)：显示 WinRunner 的可用菜单；
- 测试工具栏(Test Toolbar)：显示运行测试时可用的命令；
- 调试工具栏(Debug Toolbar)：包含脚本调试时常用的命令；
- 用户工具栏(User Toolbar)：包含录制开发测试脚本时常用的命令；
- 状态栏(Status Bar)：显示当前脚本的状态。

具体如图 16-2 所示。

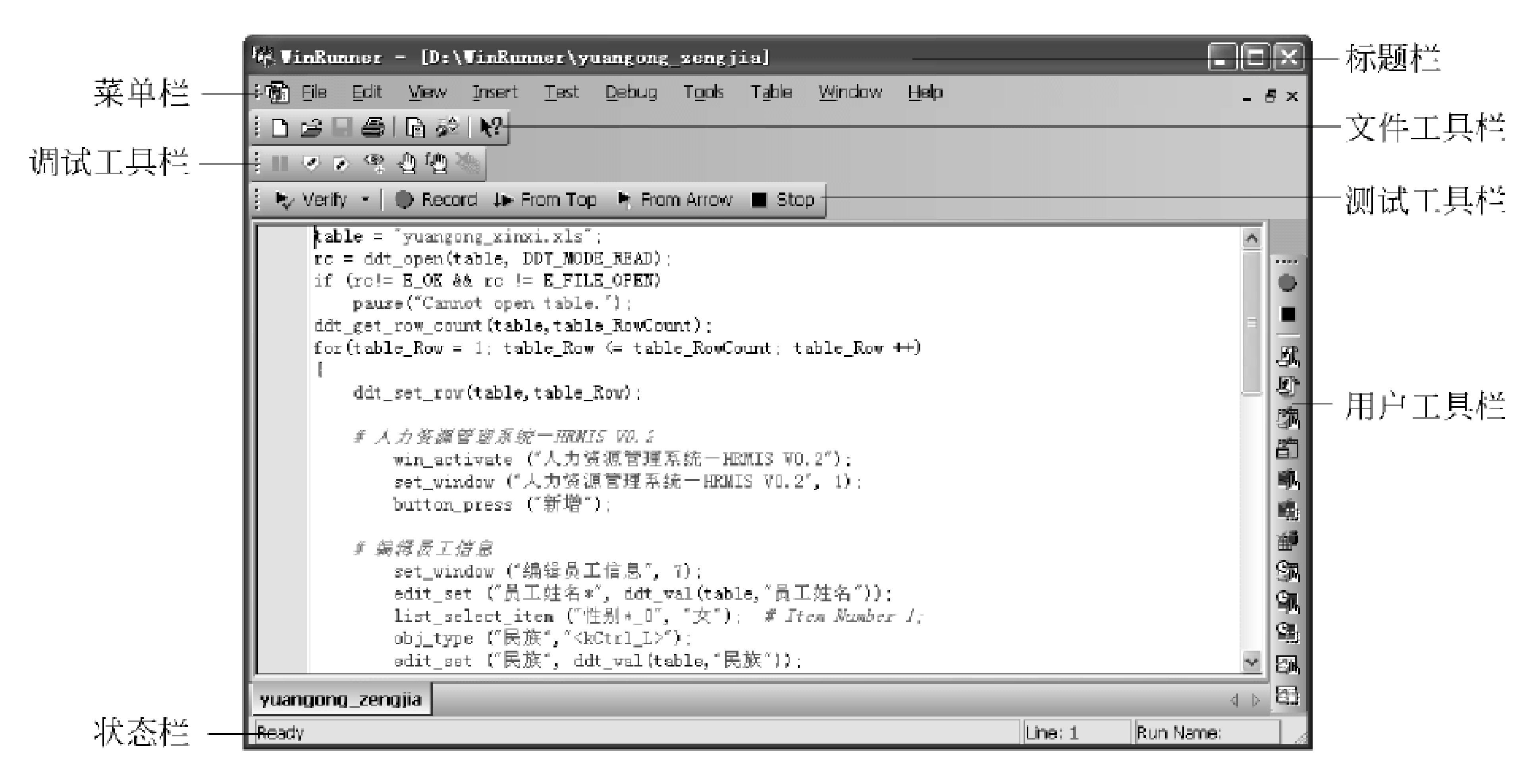


图 16-2　脚本编辑器

3）WinRunner 测试窗口

在测试窗口(如图 16-3)创建和执行测试，窗口包含以下部分。

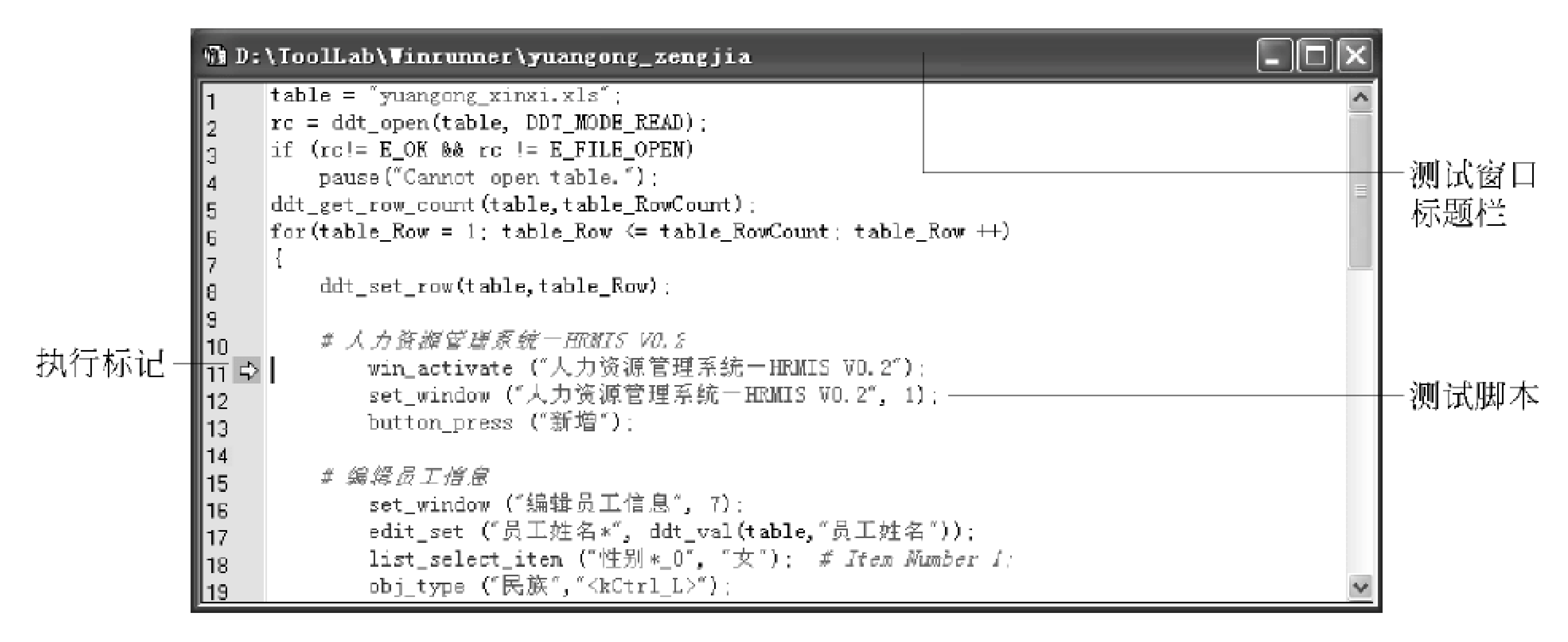


图 16-3　WinRunner 测试窗口

- 测试窗口标题栏(Test Window title bar)：标题栏显示当前打开的测试脚本名称；
- 测试脚本(Test script)：显示自动化测试脚本，脚本可以通过录制或手工编写代码的方式生成；
- 执行标记(Execution arrow)：指明当前正在执行的那一行脚本语句。如果想要

移动这个标志到某一行，只需要在该行左侧空白处点击鼠标左键。

2. WinRunner 测试流程

WinRunner 的工作流程大致可以分为以下六个步骤。

1）识别应用程序的 GUI

WinRunner 提供了一个 GUI Spy 工具，使用 GUI Spy 可以识别各种标准的 GUI 对象，如窗体、菜单、按钮等标准控件。WinRunner 会把每个 GUI 对象的描述存储到 GUI Map File 中（WinRunner 使用扩展名是.gui 的文件来保存 GUI 映像文件，在 GUI Map 中，每一个对象都有一个逻辑名称和一个物理描述，WinRunner 使用这个逻辑名称和物理描述来识别被测软件中的对象）。

WinRunner 提供两种 GUI Map File 模式：Global GUI Map File 和 GUI Map File per Test。GUI Map File per Test 对每个测试脚本产生一个 GUI 文件，能自动建立、存储、加载，这种模式不易于描述对象的改变，其效率比较低，适用于初学者；Global GUI Map File 只产生一个共享的 GUI 文件，这使得测试脚本更容易维护，且效率更高，适用于有经验的测试人员。

注意：当使用 GUI map per test 模式，可以跳过这一步骤。

2）创建测试脚本

WinRunner 提供了简单灵活的测试脚本生成方式，可以通过录制、编程或两者结合的方式开发测试脚本。为了确保程序按照预期的流程执行，我们需要检查软件是否正确响应。WinRunner 提供了检查点功能，在开发测试脚本时，可以在需要的时候（例如等待一个窗口开启、从数据库取得数据、等待状态列变为 100%、等待某个状态信息出现等情形）插入检查点（Checkpoint），来检查预期的结果是否出现。WinRunner 提供了 GUI 对象、位图（Bitmap）和数据库三种检查点方式：GUI 检查点检验 GUI 对象信息；位图检查点保存一个窗体或区域的截图，并用这张图片和以前版本进行比较；数据库检查点检查一定数量的行和列组成的数据集的内容。

录制脚本有两种模式：环境判断模式（Context Sensitive 模式）和模拟模式（Analog 模式）。环境判断模式根据选取的 GUI 对象（如窗体、清单、按钮等）把对软件的操作动作录制下来，并忽略这些对象在屏幕上的物理位置。模拟模式录制键盘的输入，鼠标的点击，和鼠标指针在屏幕上精确的 x，y 轴，记录鼠标点击、键盘输入和鼠标在二维平面上（x 轴和 y 轴）的精确运动轨迹。执行测试时，WR 让鼠标根据轨迹运动。这种模式对于那些需要追踪鼠标运动的测试非常有用，例如画图软件。

3）增强测试脚本

对测试脚本进行相应的编辑，以达到各种测试的要求，如将固定值进行参数化等。在 WinRunner 中有专门一个 Debug Toolbar 用于调试脚本错误，可以设置断点（Breakpoint）来控制和跟踪测试脚本。

4）执行测试脚本

WinRunner 提供三种脚本运行方式：验证模式（Verify mode）、调试模式（Debug mode）、更新模式（Update mode）。

调试模式：使用这种方式来帮助你识别测试脚本中的缺陷。

验证模式：当前数据和前期捕捉的期望值进行比较，该模式为默认的模式。在验证模式下测试软件，WinRunner 在脚本运行中遇到检查点后，就把当前数据和前期捕捉的期望值进行比较，并记录不一致的情况。

更新模式：使用这种方式来更新测试脚本的期望结果，为 GUI 检查点或位图检查点创建一个新的预期结果。

5）分析测试结果

当运行完某个测试脚本后，会产生一个测试报告，报告会详细描述测试执行过程中发生的所有主要事件，如检查点、错误信息、系统信息或用户信息。从这个测试报告中我们能发现应用程序的功能性缺陷，能看到实际结果和期望结果之间的差异，以及在测试过程中产生的各类对话框信息等。

6）整理报告缺陷

在分析完测试报告后，按照测试流程要汇报应用程序的各种缺陷，然后将这些缺陷发给指定人，以便进行修改和维护。

3．WinRunner 举例

本节通过 WinRunner 自带的飞机订票的例子，来讲述如何利用 WinRunner 进行自动化功能测试。

1）录制测试脚本

首先，启动飞机订票系统，然后启动 WinRunner，并设置 GUI Map File 的模式为 GUI Map File per Test，脚本录制模式选择 Context Sensitive。单击录制按钮，并切换到飞机订票界面，开始飞机订票操作（如图 16-4 所示）。

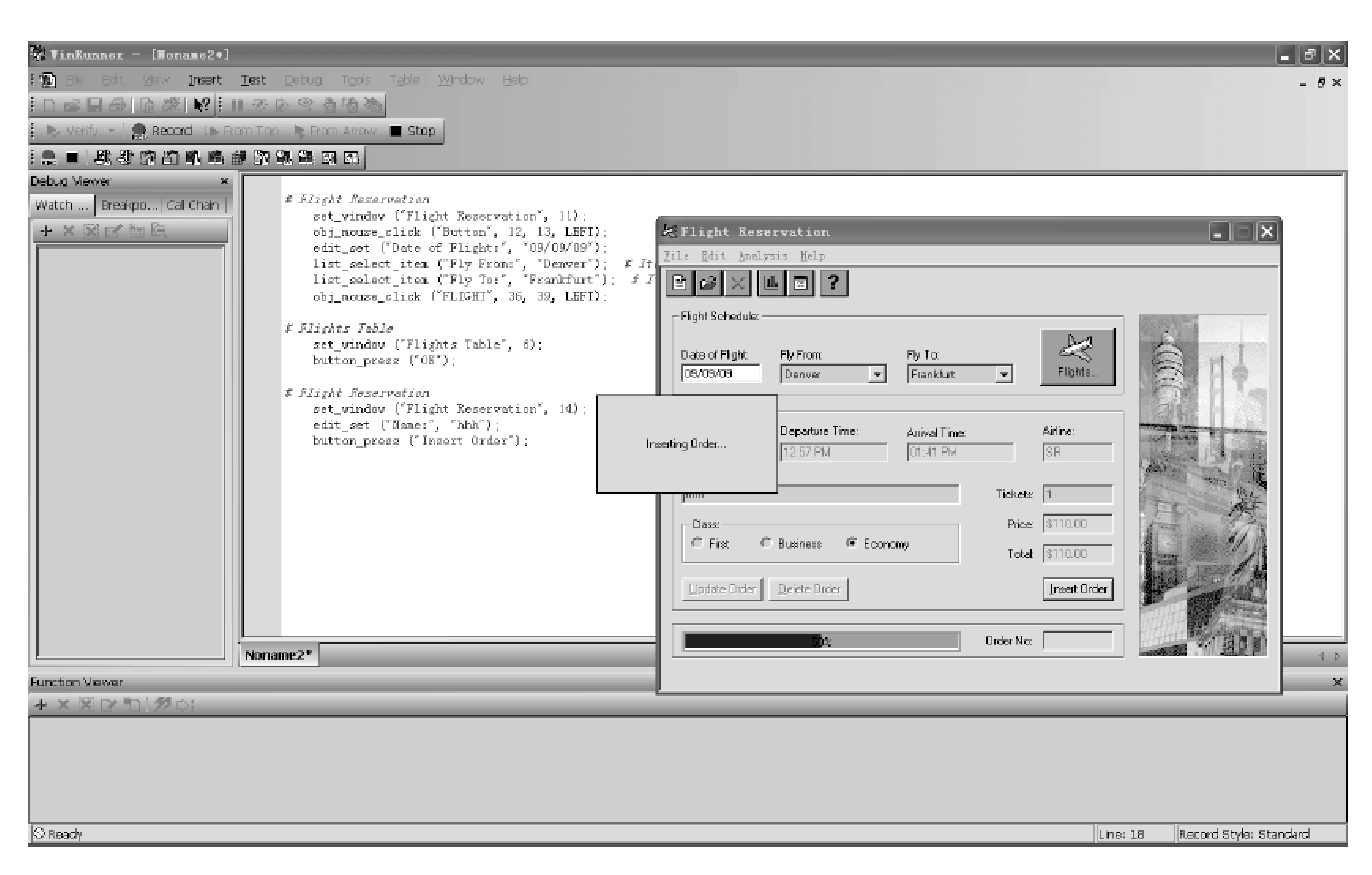


图 16-4　录制脚本

录制脚本过程中，可以加入一些检查点，来验证脚本执行是否正确，这里我们在订票完成后，加入检查点 Insert Down。在 WinRunner 界面中，依次选择 Insert→GUI

checkpoint→For Object/Window,然后选择要检查的内容(如图 16-5 所示)。

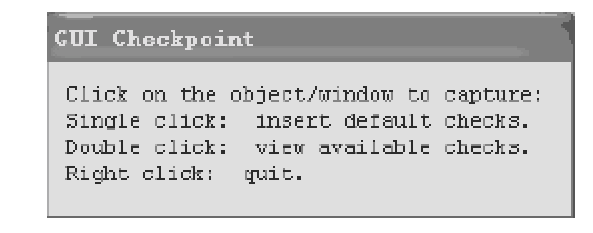




图 16-5　插入检查点

操作完毕,单击 WinRunner 工具条中的停止按钮,完成测试脚本的录制。

2) 增强测试脚本

测试脚本录制完毕,对测试脚本进行增强,以达到实际的测试需要。这里我们对订票的用户名进行参数化,参数化脚本的方法如下:在 WinRunner 的菜单中依次选择 Table→Data Table Wizard,按照数据表向导的提示单击 Next 按钮,选择要进行参数化的值设定相应的参数。数据映射关系(参数化)设置完毕,在向导的最后完成界面选择 Show data table now,打开数据表输入测试数据并保存(如图 16-6~16-7 所示)。



图 16-6　Show data table now

3) 执行测试脚本

开始执行准备好的测试结果,单击 From Top,从脚本第一行开始执行测试脚本(如图 16-8 所示)。

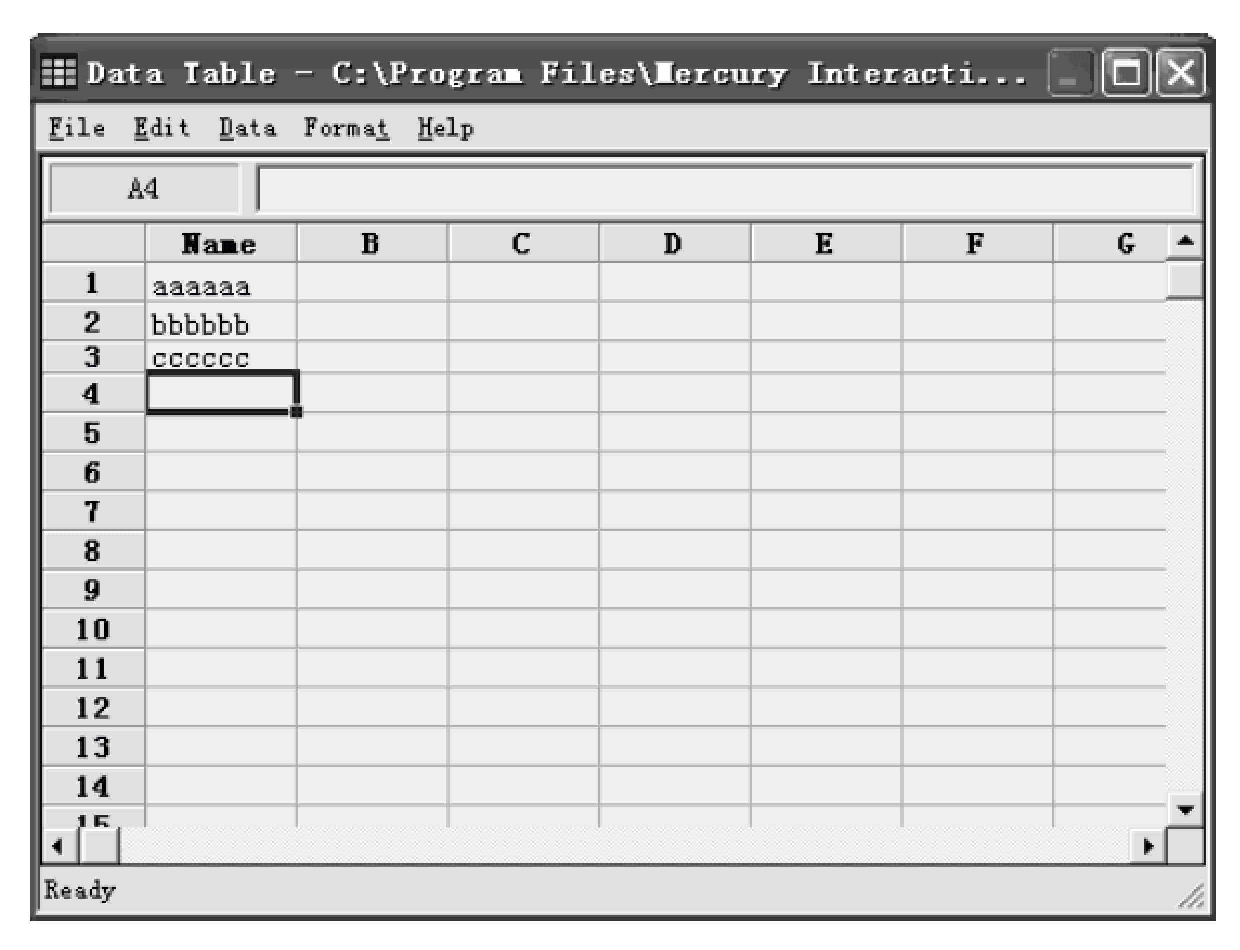


图 16-7 测试数据

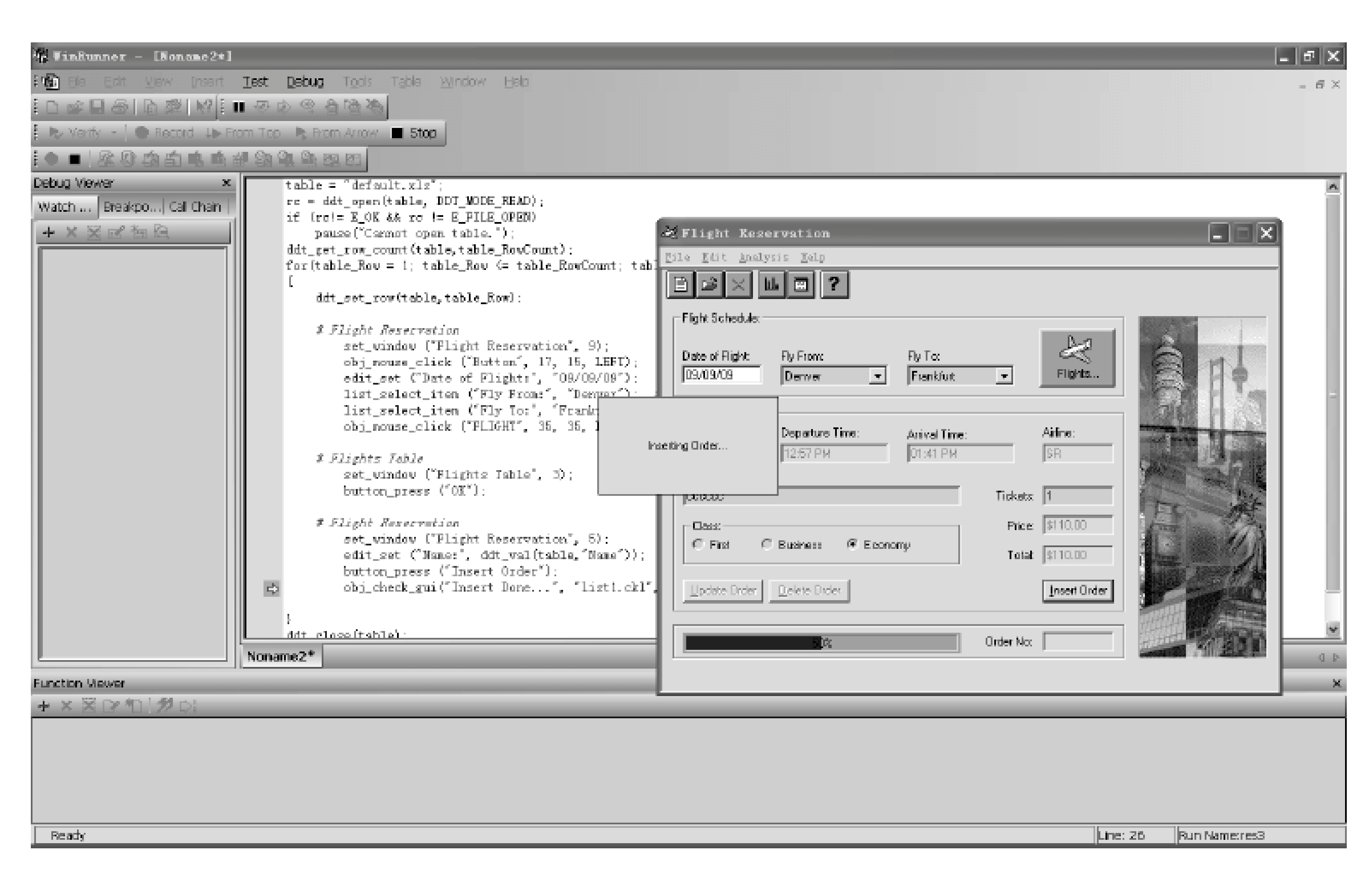


图 16-8 脚本执行

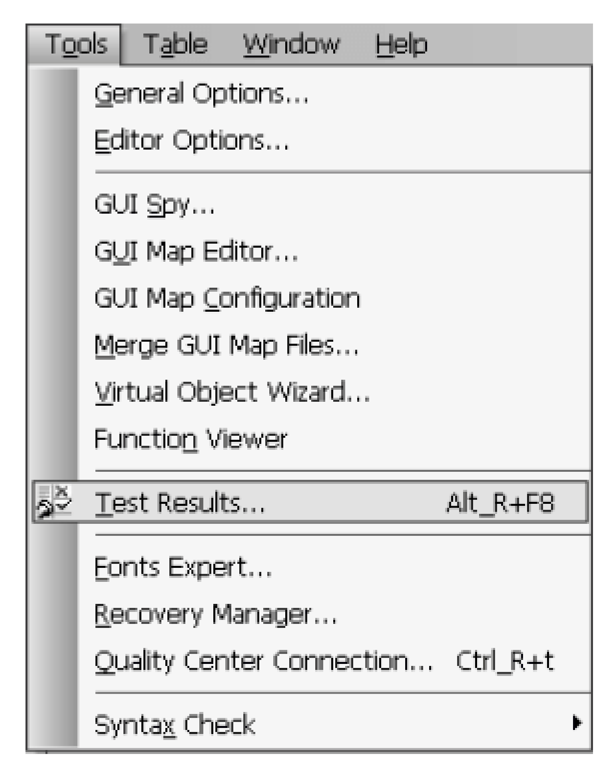


图 16-9 测试结果菜单

4）分析测试结果

测试脚本执行完成后，可以通过 WinRunner 的 Tools 菜单下的 Test Results（如图 16-9 所示）打开测试结果（如图 16-10 所示）。

从图 16-10 中我们可以得到如下信息：测试脚本执行结果为 Pass，脚本中所设置的检查点检查通过，没有发现异常。另外，在 General Infomation 栏可以看到有关脚本执行的一些信息，如执行时间、测试人、脚本运行时间等。

在 Test Results 栏，我们双击检查点的检查结果可以打开 GUI Checkpoint Results 窗口，在这里可以看到检查点的

预期结果和实际结果(见图 16-11),从这里似乎也可以感受到测试用例设计的一些思想。

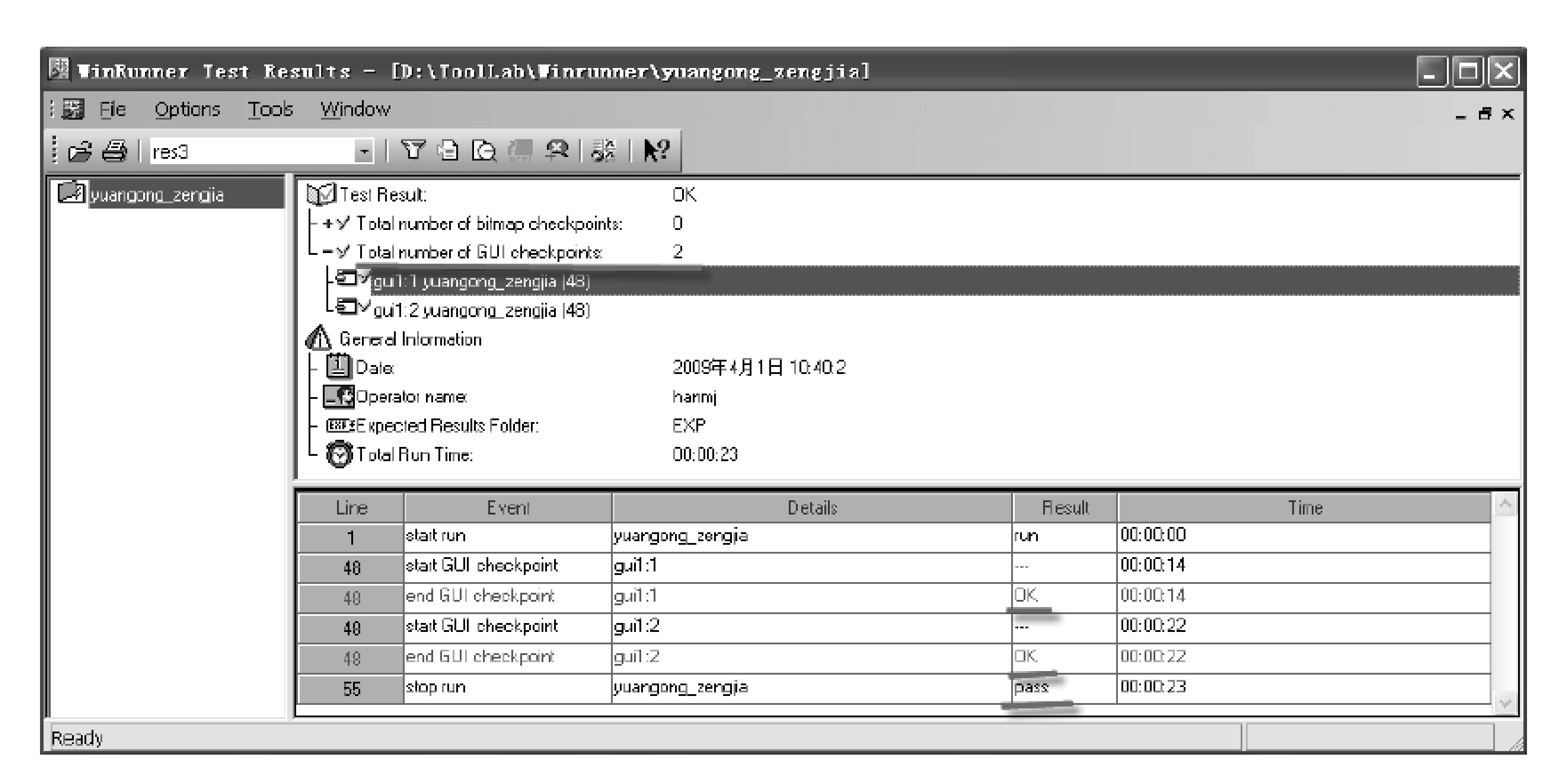


图 16-10　测试结果

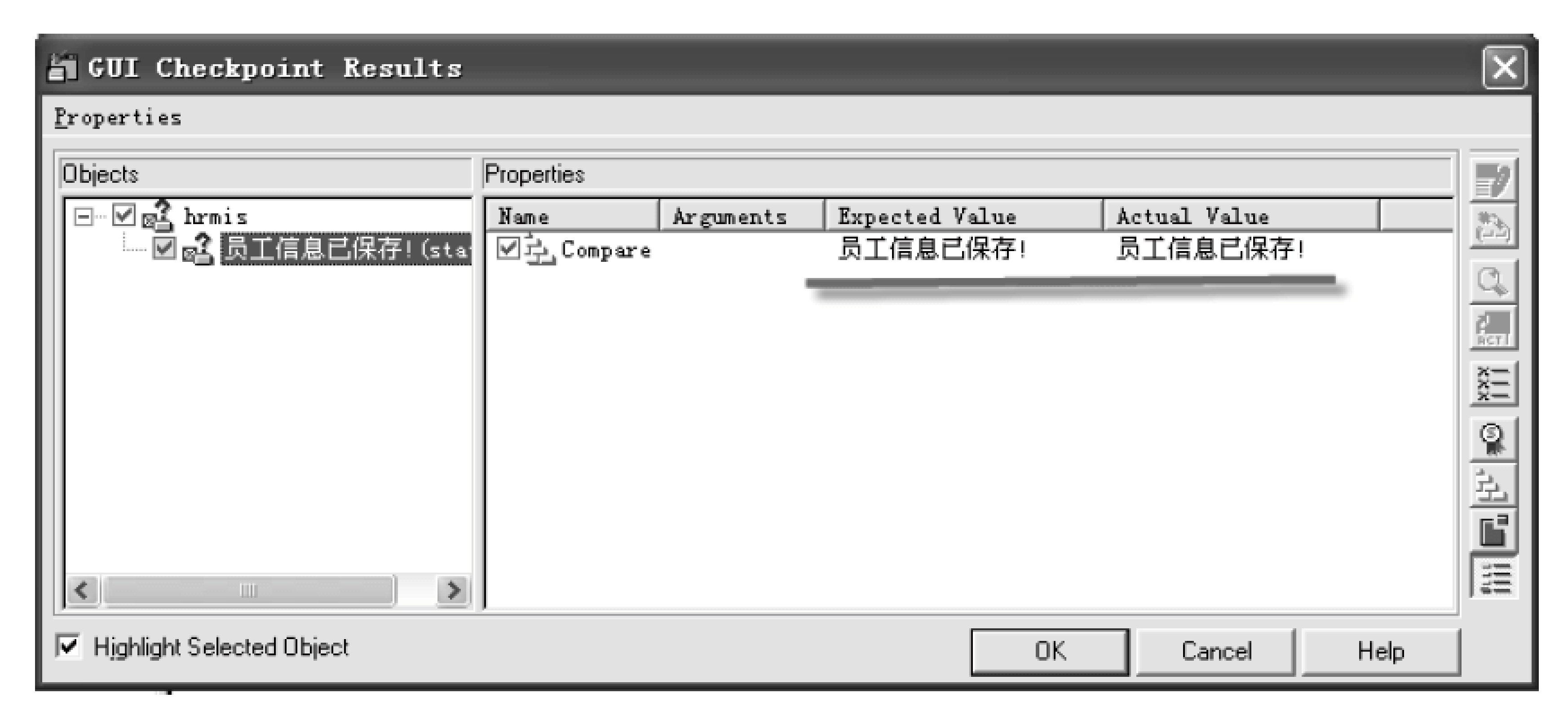


图 16-11　GUI Checkpoint Results

16.1.3　常见问题解答

1. 不能录制正确的 TSL 语句

当 WinRunner 不能正确识别对象时,就会录制成 obj_mouse 语句。可能的原因为没有加载支持对象的插件,或者对象是一个子定义类的对象等,对应的解决方法如下:

- 安装并加载支持目标对象的插件;
- 把这个自定义的类映射到那个标准类;
- 可以添加一个自定义 GUI 对象类。

2. 不能从 HTML 页面读取文本

当遇到 WinRunner 不能从 HTML 页面中读取文本时,可能的原因为没有安装并加载支持 Web 对象的插件,或者 WinRunner 不能识别 HTML 框架或表格中的文本,对应的解决方法如下:

(1) 安装并加载支持 Web 对象的插件。

(2) 选择 Create→Get Text→From Selection(Web only)命令从 HTML 页面中重新得到文本。

(3) 选择 Create→Get Text→WebText Checkpoint 命令检查是否有特定的字符串存在于 HTML 页面中。

3. WinRunner 是否支持 vs.net

根据 HP 介绍,WinRunner 对.Net 的支持转移到 QuickTest Professional 上,如果需要自动化测试.Net 程序,建议用 QuickTest Professional。

16.2 Quick Test Professional

16.2.1 概述

QTP(Quick Test Professional)是 HP 公司继 WinRunner 产品之后推出的以 VB Script 为内嵌语言的一款先进的自动化功能测试工具。QTP 针对的是 GUI 应用程序,包括传统的 Windows 应用程序,以及现在越来越流行的 Web 应用。QTP 可以覆盖绝大多数的软件开发技术,简单高效,并具备测试用例可重用的特点,用于创建功能和回归测试。

QTP 主要具有如下功能特点:

① QTP 可以轻松创建测试脚本,快速完成功能点的测试。只需通过单击录制按钮,按照典型的业务流程对应用程序进行操作,即可创建测试脚本。QTP 使用简明的语句和屏幕抓图来自动记录业务流程中的每个步骤,可以使用户在关键视图中修改、删除测试脚本或重新安排测试步骤变得更加轻松,提高测试效率。

② QTP 可以建立共享对象库,供整个测试团队使用同一测试脚本,从而消除重复工作,同时当应用程序出现变动时,如“登录”按钮的名称改为“签到”等,只需修改共享对象库,即可使整个团队直接使用原来的测试脚本,大大提高了工作效率。

③ QTP 同时满足了技术型和非技术型用户的需求,让各个公司有能力部署更高质量的应用,同时部署的速度更快,费用更低,风险也更小。QTP 可以将非技术型的业务专家引入质量流程,可以将 IT 和业务更好地融合,创建更出色的应用程序。

④ QTP 可以独立运行,也可以同其他测试管理工具相结合,如 Test Director 等,使得测试文档更容易管理,甚至可以使用 TD 来调用 QTP 进行测试脚本录制(在本机上)。

⑤ QTP 使用范围广泛,支持所有常用环境的功能测试,包括 Windows、Web、.NET、Visual Basic、ActiveX、Java、SAP、Siebel、Oracle、PeopleSoft 和终端模拟器等。

⑥ QTP 具有自动文档技术,可以实现测试文档的建立与测试脚本的建立同步。

⑦ QTP 可以提供详细全面的测试结果。测试报告记录测试运行的所有方面,准确指出应用程序故障位置的可扩展树视图,使用的测试数据,突出显示任何差异的应用程序屏幕抓图,以及每个通过和未通过检查点的详细说明。

16.2.2 Quick Test Professional 的应用

1. 使用概述

1) QTP 启动

选择“开始”→“程序”→Quick Test Professional→Quick Test Professional 启动 QTP,首先出现加载插件窗口(如图 16-12 所示)。根据被测试软件,选择插件类型,单击 OK 按钮,启动 QTP。

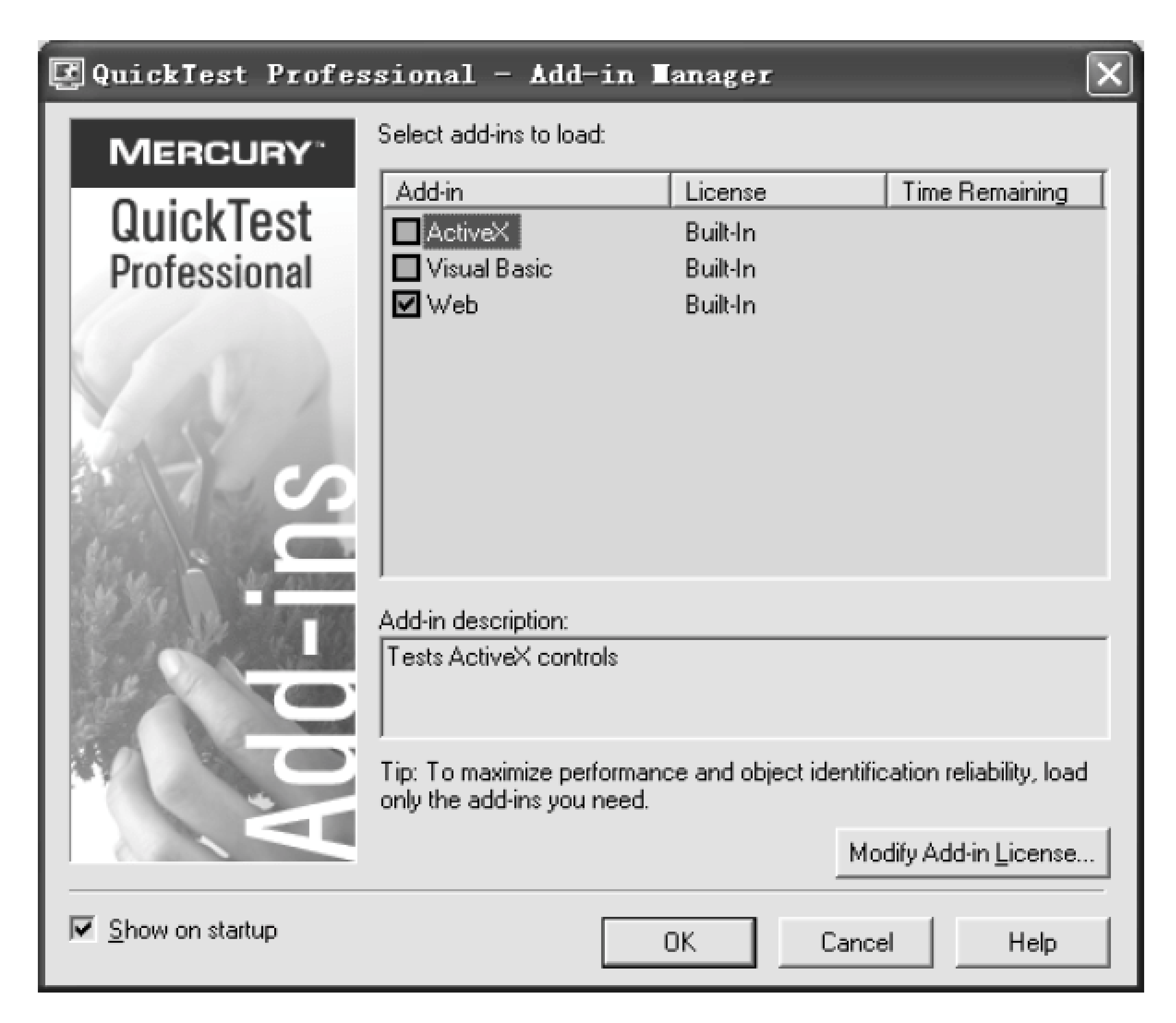


图 16-12 启动 QTP

2) QTP 主窗口

QTP 的主窗口包括以下部分(如图 16-13 所示):

• Title Bar:显示目前测试脚本的名称;

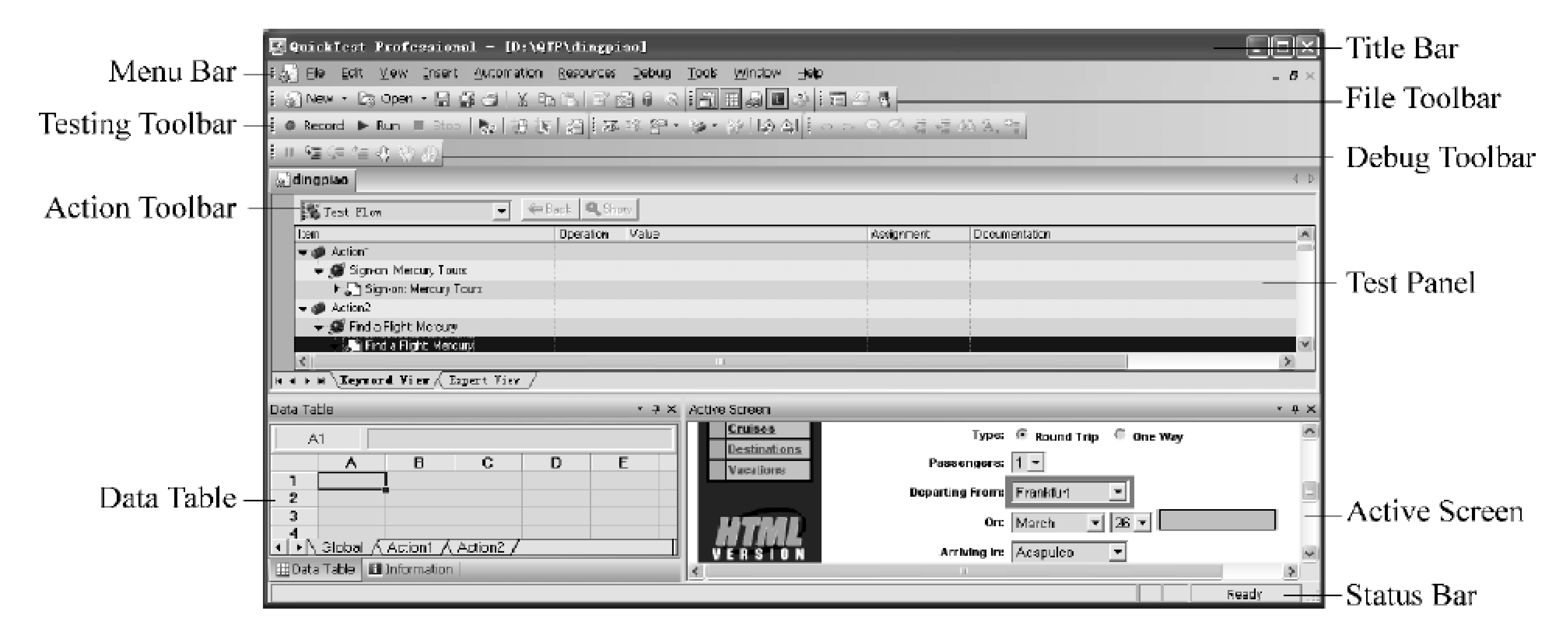


图 16-13 QTP 主窗口

- Menu Bar：显示 QuickTest 的菜单；
- File Toolbar：管理测试脚本常用的工具列；
- Testing Toolbar：录制测试脚本常用的工具列；
- Debug Toolbar：调试脚本常用的工具列；
- Action Toolbar：包含常用的功能按钮，以及一个显示测试动作的下拉式清单，方便检视整个测试脚本中的测试动作；
- Test Panel：显示测试脚本，包含 Keyword View 以及 Expert View 两种模式；
- Active Screen：测试脚本当前访问的活动窗口；
- Data Table：存放测试脚本参数化数据；
- Debug Viewer panel：协助对测试脚本除错；
- Status Bar：显示测试脚本的状态。

2. QTP 测试流程

1）录制测试脚本

输入被测软件的地址或路径，并对被测软件进行相应的操作，QTP 会以表格的方式显示录制的操作步骤，每一个操作步骤都是使用者在录制时执行的操作，如单击一个超级链接或图像，或是向表单提交数据等。

2）增强测试脚本

录制完毕测试脚本，需要对测试脚本进行增强，以达到测试的要求。对测试脚本进行参数化、加入检查点、同步点等操作，或者添加逻辑和条件语句或循环语句，来实现更复杂的测试。

3）调试测试脚本

调试测试脚本，确保测试可以流畅而无中断地运行。

4）运行测试脚本

运行测试准备的测试脚本，模拟实际的各种操作情况，并得到测试结果。

5）分析测试结果

检查测试结果，以便确定应用程序中的缺陷。

6）报告缺陷

将发现的缺陷发给指定人，以便进行修改和维护。

3. QTP 举例

本节通过 QTP 自带的飞机订票的例子来讲述 QTP 的使用。

1）录制脚本

首先，启动 QTP，选择 Web 插件(见图 16-14)然后选择新建一个测试，指定录制的网站地址 http://newtours.demoaut.com/(该飞机订票系统随 QTP 一同安装，可以直接使用)，开始录制脚本(如图 16-13 所示)，输入用户名和密码(可以单击首页的 REGISTER，先注册一个用户名)，进入订票页面，选择相关选项。

由于本测试脚本是用来测试订票功能，而不是考察系统登录功能，所以需要将系统登录部分和订票操作部分分开处理，以达到模拟用户登录系统后反复执行订票操作的要求。

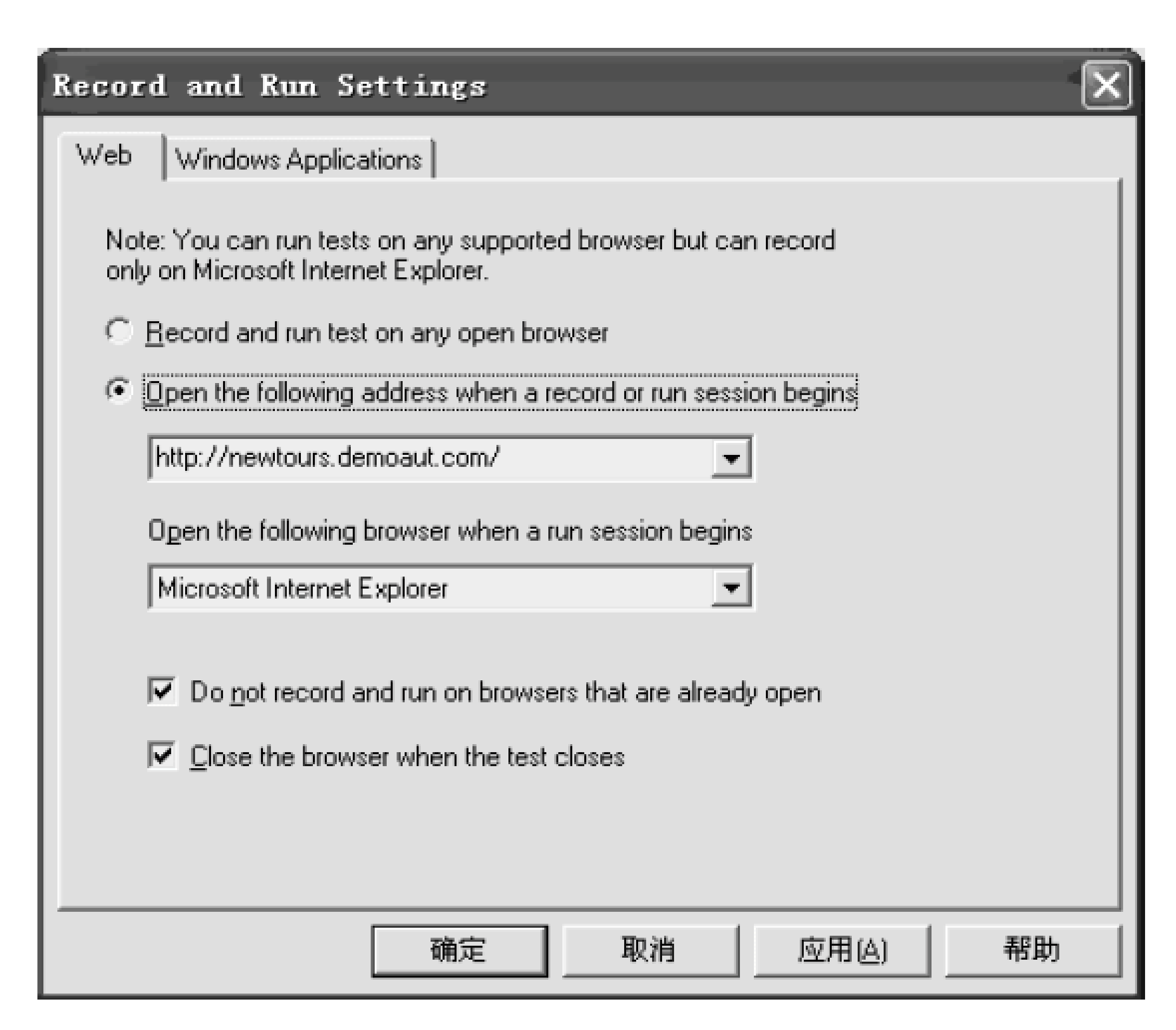


图 16-14　录制脚本

可以在脚本录制过程中,将系统登录部分和订票操作部分分别录制在不同的 Action 中,如图 16-15 所示,系统登录部分脚本录制完毕后,单击 QTP 的 Testing Toolbar 中的 Insert Call to New Action,插入新的 Action2,开始订票操作部分的脚本录制。

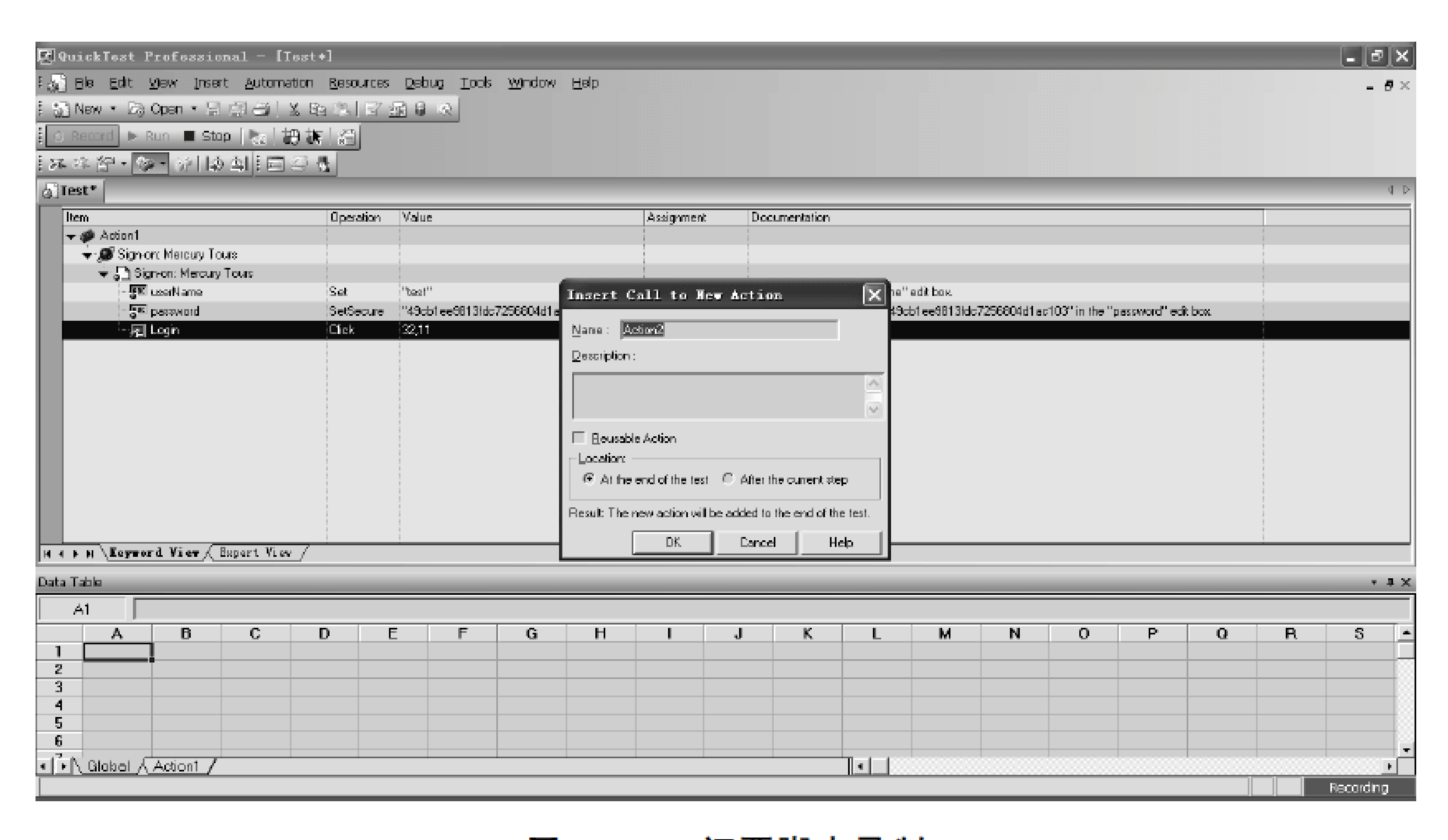


图 16-15　订票脚本录制

输入乘客的信息,姓名和信用卡号,保存订票信息。订票操作完毕,单击页面中的 BACK TO FLIGHTS,回到订票操作的初始页面(此步的目的是为了保证订票操作可以反复执行),单击 QTP 的 stop 停止录制,完成测试脚本录制(如图 16-16 所示)。

2) 增强测试脚本

(1) 检查点。

我们选择订票成功后的提示字符"Your itinerary has been booked!"作为检查点,选

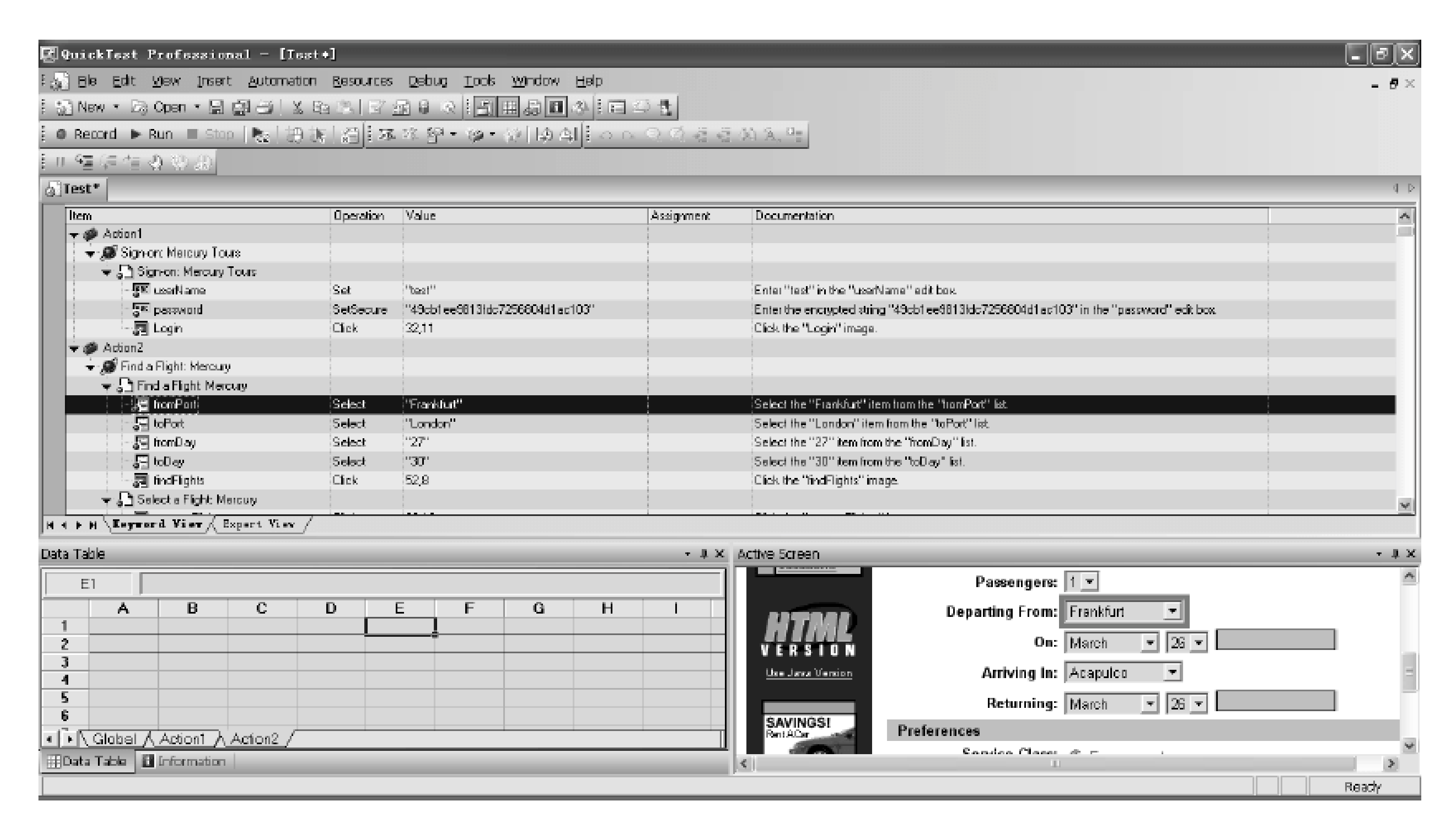


图 16-16　订票测试脚本

择订票成功对应的语句，同时 Action Screen 会显示对应窗口(如图 16-17 所示)，选中“Your itinerary has been booked!”，单击鼠标右键，在弹出菜单中选择 Insert Text Checkpoint：

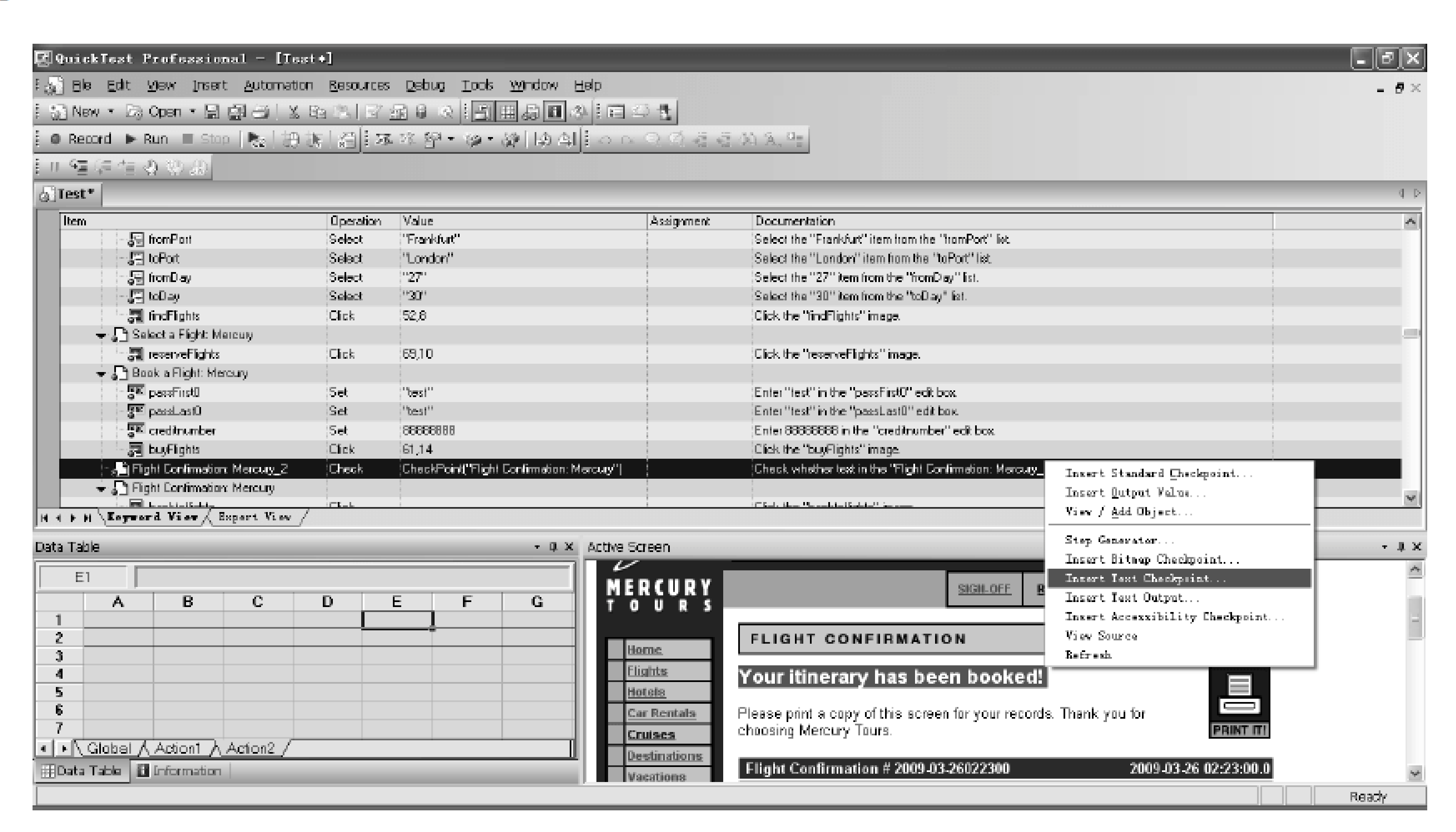


图 16-17　插入检查点

(2) 参数化。

这里只对乘客的信用卡号进行参数化。单击信用卡字段的 Value 值右边的按钮，在 Value Configuration Options 窗口中，选择 Location Data Table 中的 Current Action Sheet(local)，即只用于当前的 Action2，并在 Data Table 中输入参数(如图 16-18 所示)。

(3) 迭代。

设置 Action2 的迭代次数，即参数的选择次数。右击测试脚本中的 Action2，选择 Action Call Properties，在 Run 选项中选择参数表的选择方式，这里选择 Run on all

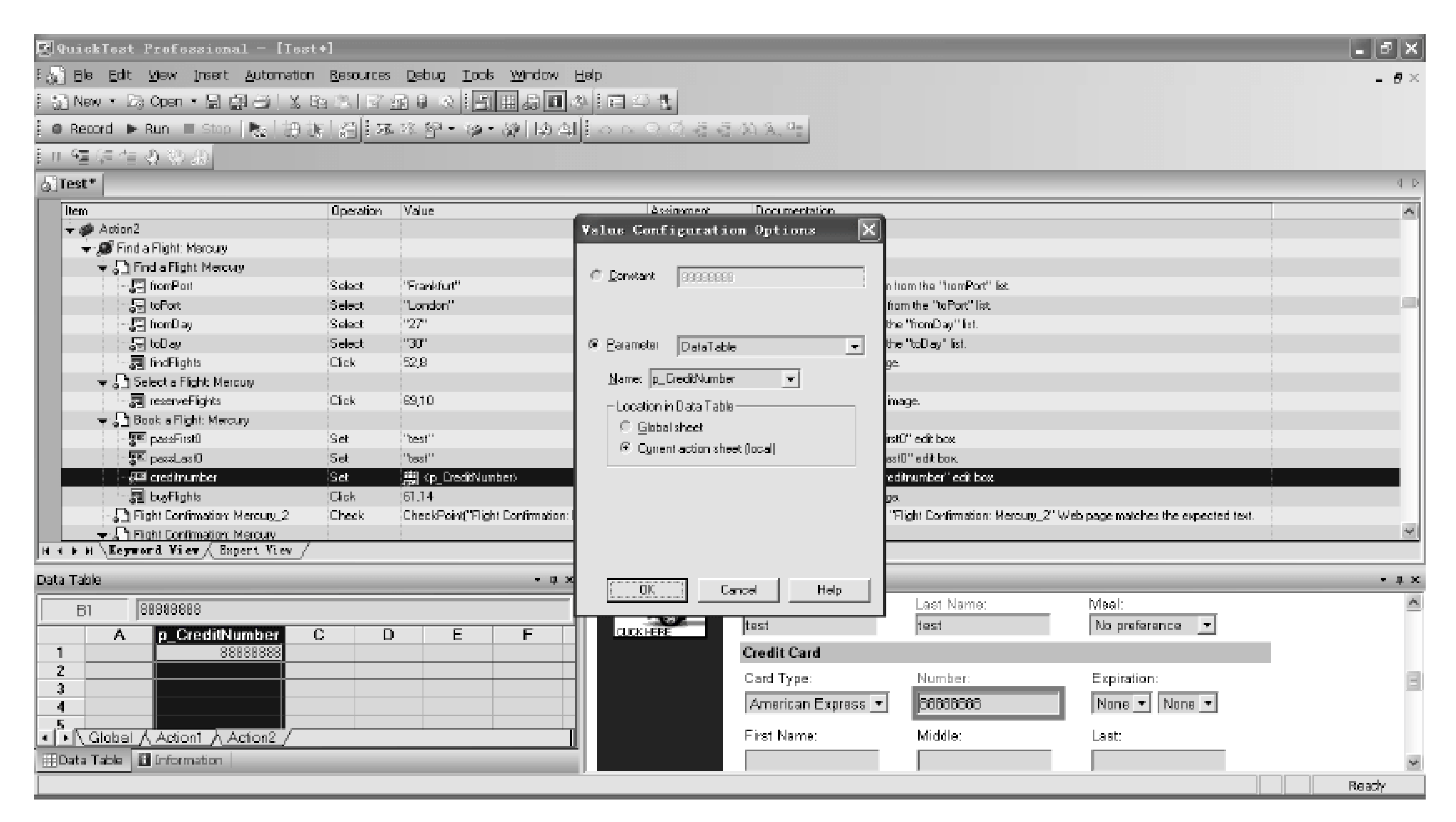


图 16-18　脚本参数化

rows,即运行参数表中的所有参数(如图 16-19 所示)。

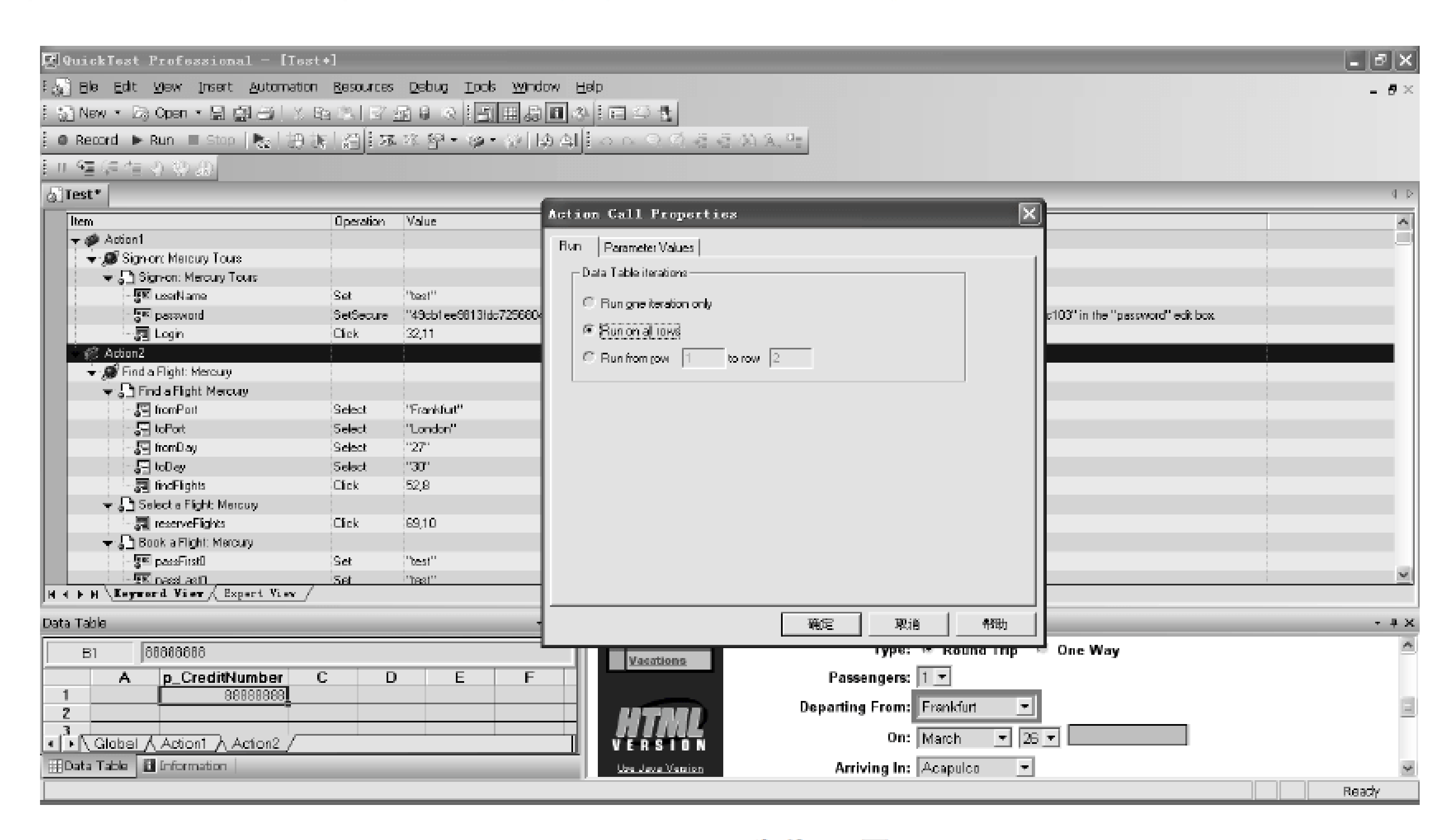


图 16-19　迭代设置

3) 调试测试脚本

通过 QTP 的 Tools 菜单中的有关命令,对脚本进行相应的调试,如单步调试、加入断点等,保证测试脚本正常运行(如图 16-20 所示)。

4) 运行测试脚本

运行测试脚本前首先设置测试结果保存目录,然后执行测试脚本。正常情况下,脚本执行完成后,可以看到如图 16-21 所示的测试结果。

5) 查看测试日志

在测试结果目录的 LOG 目录中,有对应的测试脚本运行日志文件,查看该文件内容我们可以看到更加详细的测试执行信息(见图 16-22)。

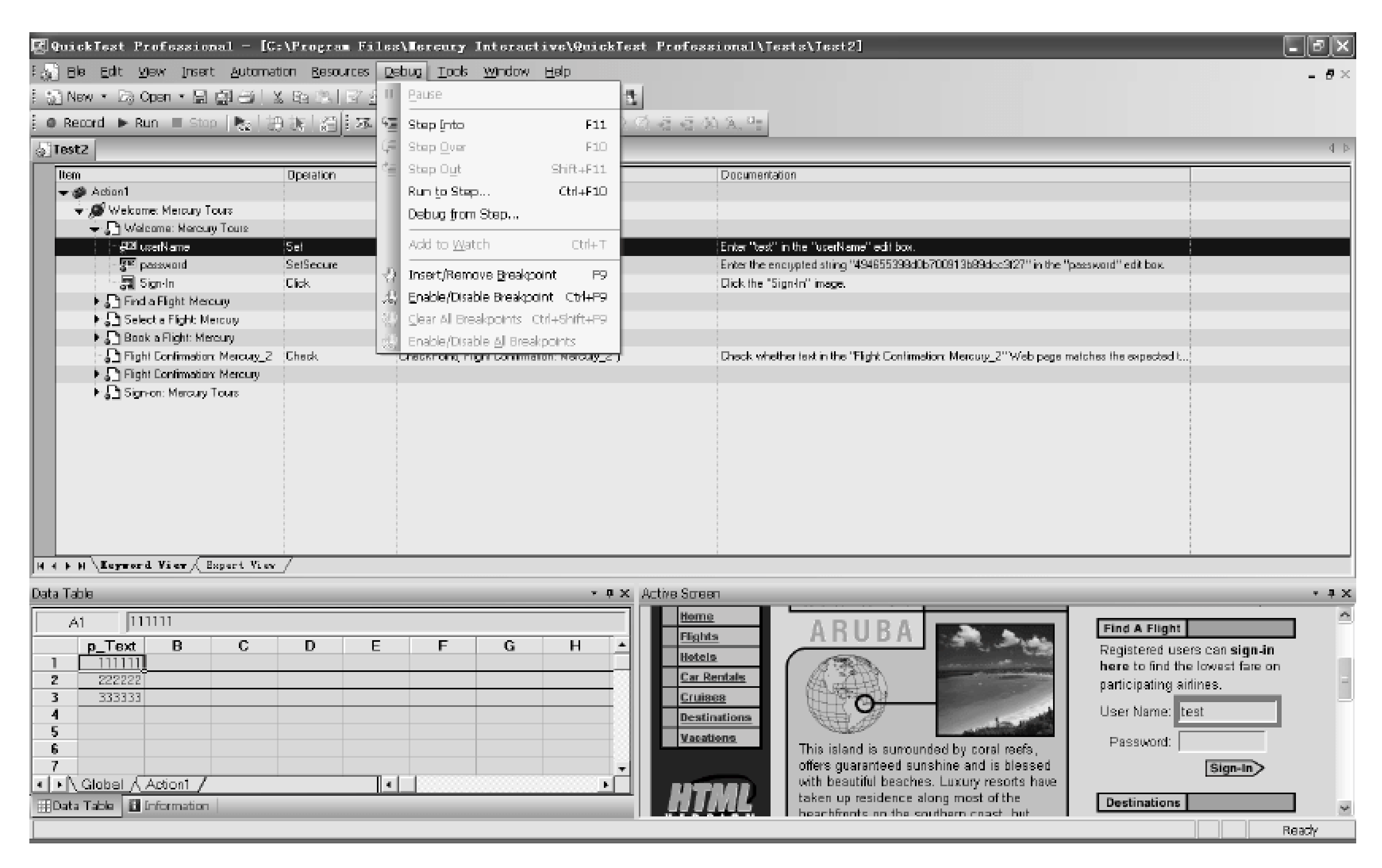


图 16-20　脚本调试

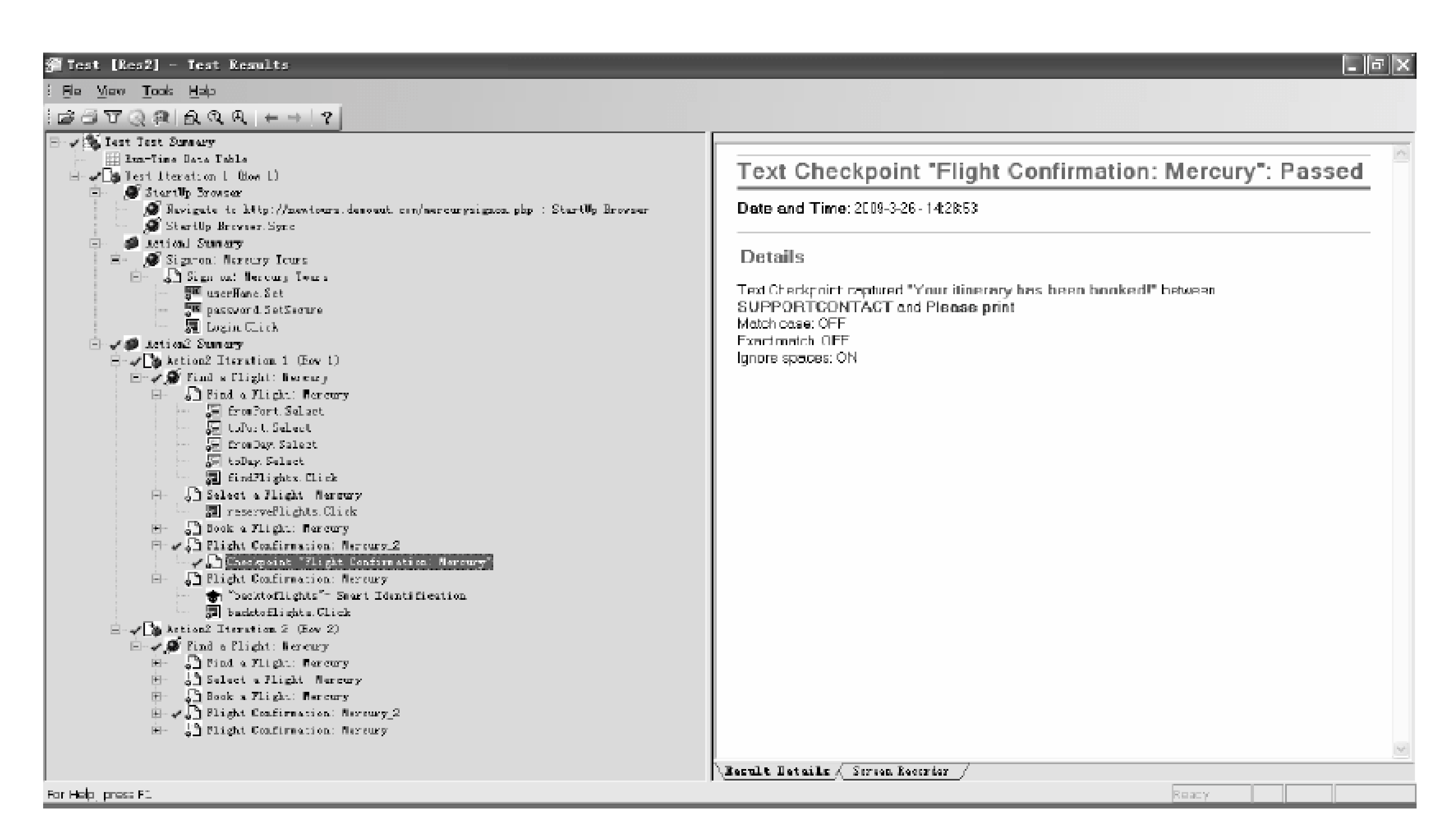
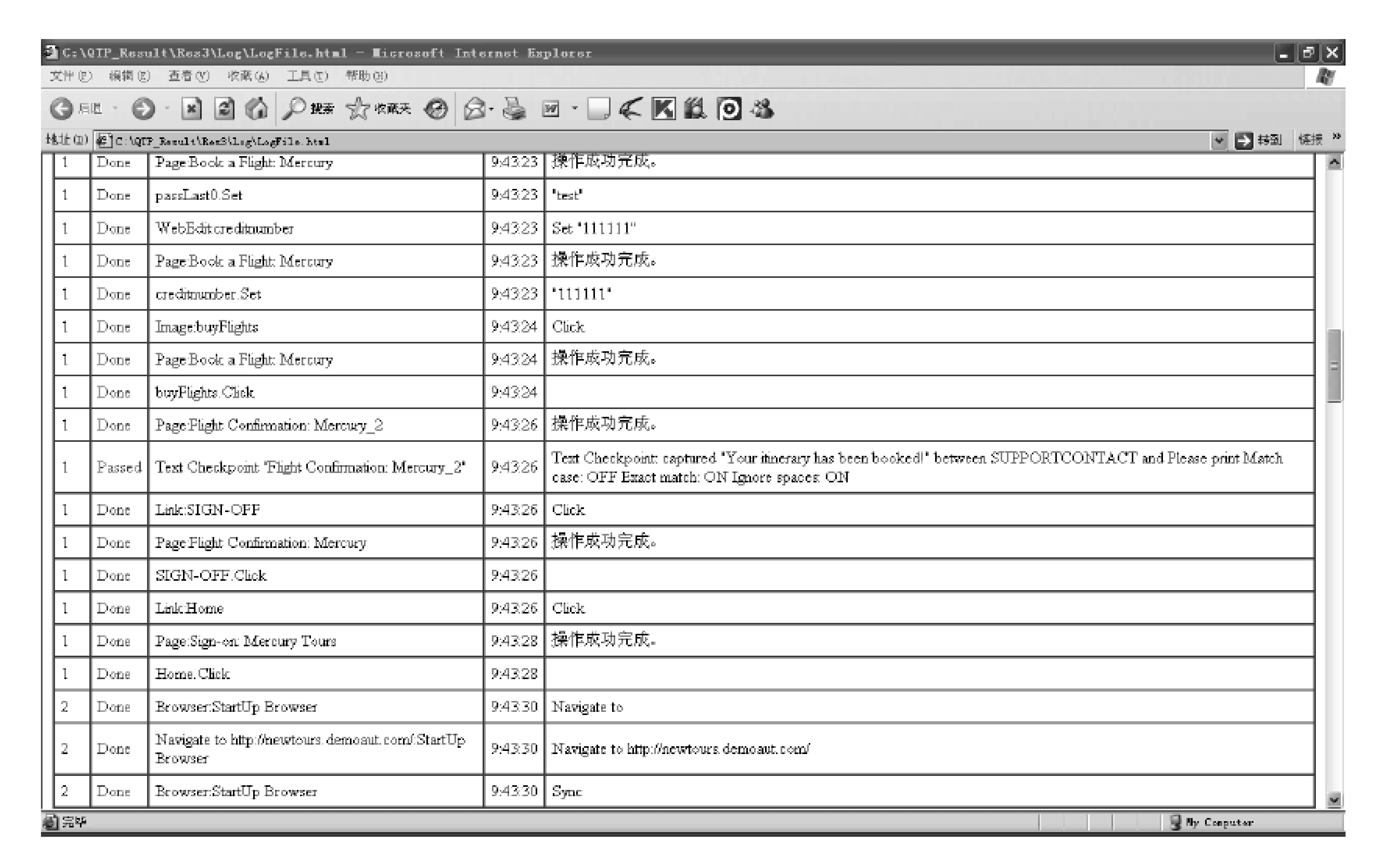


图 16-21　测试结果

小提示：QTP 生成的测试日志默认格式为 XML，查看不方便，可以通过修改注册表的相关设置，生成易于查看的 HTML 格式，该注册表项位于以下位置：

```
HKEY_LOCAL_MACHINE\SOFTWARE\Mercury Interactive\QuickTest Professional\Logger
\Media\Log
```

把其中的 Active 这一项的值修改为 1，就可以让 QTP 每次运行测试之后，在测试结果文件夹中自动产生一个名为 LOG 的目录，在里面保存一个包含测试报告的 HTML 文件。



1	Done	Page:Book a Flight: Mercury	9:43:23	操作成功完成。
1	Done	passLast0.Set	9:43:23	"test"
1	Done	WebEdit:creditnumber	9:43:23	Set "111111"
1	Done	Page:Book a Flight: Mercury	9:43:23	操作成功完成。
1	Done	creditnumber.Set	9:43:23	"111111"
1	Done	Image:buyFlights	9:43:24	Click
1	Done	Page:Book a Flight: Mercury	9:43:24	操作成功完成。
1	Done	buyFlights.Click	9:43:24	
1	Done	Page:Flight Confirmation: Mercury_2	9:43:26	操作成功完成。
1	Passed	Text Checkpoint:"Flight Confirmation: Mercury_2"	9:43:26	Text Checkpoint: captured "Your itinerary has been booked!" between SUPPORTCONTACT and Please print Match case: OFF Exact match: ON Ignore spaces: ON
1	Done	Link:SIGN-OFF	9:43:26	Click
1	Done	Page:Flight Confirmation: Mercury	9:43:26	操作成功完成。
1	Done	SIGN-OFF.Click	9:43:26	
1	Done	Link:Home	9:43:26	Click
1	Done	Page:Sign-on: Mercury Tours	9:43:28	操作成功完成。
1	Done	Home.Click	9:43:28	
2	Done	Browser:StartUp Browser	9:43:30	Navigate to
2	Done	Navigate to http://newtours.demoaut.com/.StartUp Browser	9:43:30	Navigate to http://newtours.demoaut.com/
2	Done	Browser:StartUp Browser	9:43:30	Sync

图 16-22　测试日志

16.2.3　常见问题解答

1. 浏览器问题

1）IE 浏览器自动完成功能

假如使用的是 IE 浏览器，则需要取消 IE 浏览器的自动完成功能，不让浏览器自动记录账号和密码，以避免录制脚本过程中，所有的操作过程都被记录下来。方法：

开启 IE 浏览器，选择“工具”→“Internet 选项”→“内容”→“自动完成”命令，取消“表单上的用户名和密码”选项，单击“确定”按钮（如图 16-23 所示）。

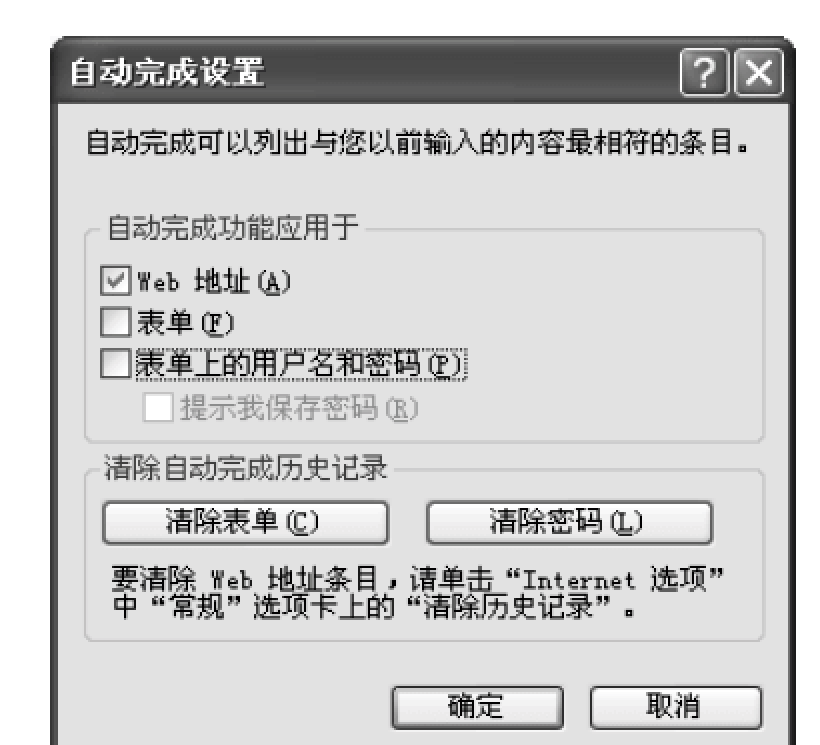


图 16-23　自动完成

2）浏览器插件问题

在运行 QTP 时，由于 QTP 对浏览器的反应速度比较高，一些影响浏览器浏览速度的插件将导致 QTP 无法正常运行。对于该类问题，在录制脚本前，删除浏览器插件。

2. 检查点设置问题

QTP 通过检查点可以检查流程是否正确，但如果该检查点既需要检查正确的情况，又需要检查不正确的情况时，这时候需要分开录制脚本。以一个系统的登录操作为例，就需要录制两个脚本，分别对应登录成功和登录失败，需要在各自的脚本中设置不同的检查点、不同的参数化。

在建立文字检查点后，参数化检查点，但是随着每次循环环境的不同，检查点可能无

法正确检查到文字，这种情况是因为检查点两边的限制条件发生变化的原因造成的，解决这类问题，需要对文字检查点的限制条件也进行参数化。

3．虚拟对象问题

在实际测试过程中，很多时候网页上的控件并不能都被 QTP 识别，例如网页上一个 Flash 动画，实际上是一个链接，但是 QTP 无法识别出 Flash 这个控件。QTP 提供一种虚拟对象的功能来解决这个问题，将不能识别的对象虚拟成 QTP 可以识别的对象。选择 QTP 菜单中的 Tools→Virtual Objects→New Objects 命令，启动虚拟对象向导，根据向导指示，完成虚拟对象的操作。

4．Action Screen 捕捉不精确的问题

QTP 不能 100％精确地捕捉到 Action screen，但可以通过 QTP Action Screen 的设置来逐步精确(建议在录制时设检查点，避免捕捉不精确的问题)。在早期 QTP 菜单中，可以选择 Tools→Options→Active Screen 命令，调整录制级别，或者进行自定义级别。设定好之后，就可以录制脚本了，QTP 会根据用户的设定保存 Active Screen。

5．中文输入法的问题

由于录制和回放时输入法设定不一致，使得测试脚本回放时，QTP 不能识别某些输入法的特定字符，导致脚本无法正常运行。对于该问题的解决办法：保证录制脚本的环境与回放时的环境相同，避免产生类似的问题。

16.3 WinRunner 和 QTP 的区别

WinRunner 和 QTP 是 HP-Mercury 公司推出的两款功能自动化测试工具，在适用环境上有共同之处，也有一定的区别，具体见表 16-1。

表 16-1 WinRunner 和 QTP 的适用范围

类　别	WinRunner	QTP
Web-Related Environments	Internet Explore Netscape AOL JDK Java Foundation Classes AWT Symantec Visual Cafe ActiveX Controls	
ERP/CRM	Oracle Jinitiator，11i NCA Baan PeopleSoft Windows Siebel 5，6 GUI Clients Oracle GUI Forms	SAP Siebel 7. x PeopleSoft 8. x

续表

类　别	WinRunner	QTP
Custom Client Server	Windows C++ /C Visual Basic PowerBuilder Forte Delphi Centura Stingray SmallTalk	
Operating Systems	Windows 98/2000/NT/Me/XP	
Legacy	3270,5250 Emulators VT100	
NET Web Services Multimedia		WinForm WebForms . NET controls XML HTTP WSDL SOAP J2EE . NET RealAudio/Video Flash

本章小结

本章主要介绍了 HP 公司的两款典型的功能测试工具 WinRunner 和 QTP,讲解了 WinRunner 和 QTP 的功能特点、使用流程和常见问题的一些解决办法,另外为了加深理解我们以工具自带的飞机订票系统为例,具体示范了 WinRunner 和 QTP 的应用过程。

CHAPTER

第 17 章 性能测试工具

性能测试目前已成为软件测试的一个热点领域。性能的范畴非常广泛，比如软件的交易效率、服务器的资源利用率、可靠性、稳定性等均可以视为性能的一个方面。本章将向大家介绍几款性能测试工具，通过工具的介绍和应用以更好地理解性能测试。

17.1 性能测试概述

通常讲的这个性能测试指的是通过专业的测试工具，模拟多种负载条件下的使用场景，监控记录各种交易执行指标和资源消耗情况，用以评估系统的性能状况。常见的性能测试类型有负载测试、压力测试和最大工作量强度测试(疲劳测试)。

从性能测试的定义可以看出，手工方式是无法满足实施性能测试的要求的，所以性能测试的自动化程度非常高，在性能测试的实施过程中，人工干预的成分相对来说比较少。

性能测试的结果本质上反映的是软件六大质量特性的效率特性，通过各种类型的性能测试可回答如下一些问题：

- 应用程序是否能够很快地响应用户的要求?
- 应用程序是否能处理预期的用户负载?
- 应用程序是否能处理业务所需的事务数量?
- 在预期和非预期的用户负载下，应用程序是否能够稳定运行?

17.1.1 常见的软件性能指标

准确理解和掌握软件的性能指标是做好性能测试的基础，软件性能指标是测试结果分析的依据，是反映软件性能状况的重要数据。下面让我们来认识一下几个常见的软件性能指标，它们是响应时间、并发用户数、吞吐量、交易成功率和资源利用率。

1. 响应时间

响应时间指的是客户端发出请求到得到响应的整个过程所经历的时间。如图 17-1

所示为典型的三层架构的系统一次软件操作的响应过程,其中标示以 N 开头的为网络传输时间,以 A 开头的为服务器处理时间。

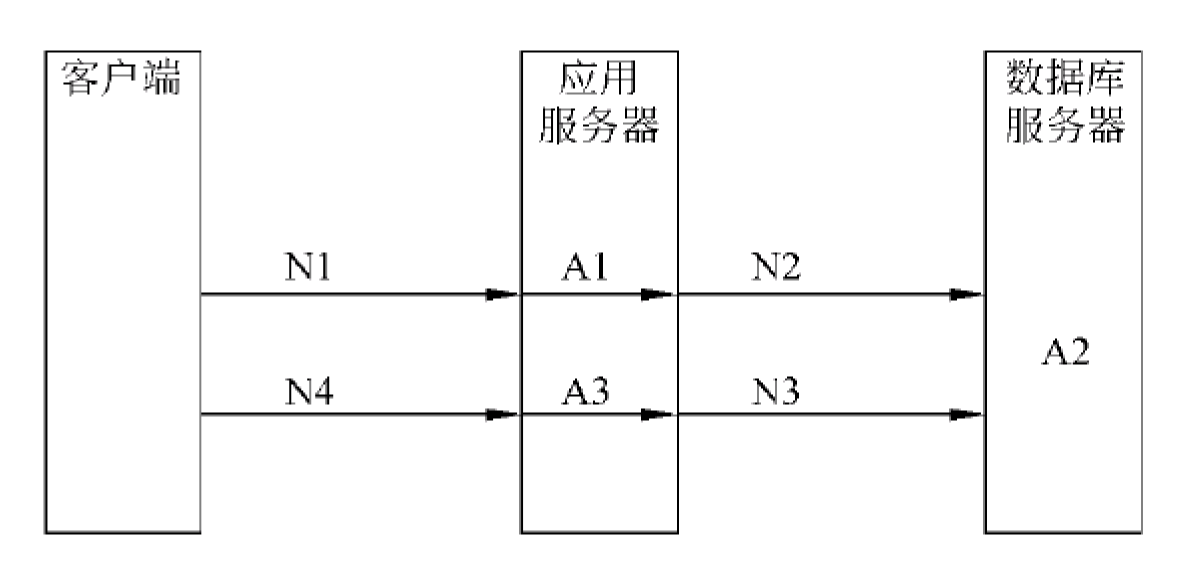


图 17-1　分段响应时间示意

2. 并发用户数

并发用户数指在某个时间特定点上与服务器端进行会话操作的用户数。有狭义的并发用户数和广义的并发用户数:

- 狭义的并发用户数是指多个用户并发执行同一操作,如若干用户同时单击查询按钮,或者同时单击发送按钮等。
- 广义的并发用户数是指多个用户同时执行不同的操作,如若干用户并发执行查询操作,同时若干用户执行发送邮件操作,还有些用户执行新增操作等。

3. 吞吐量

吞吐量是指单位时间内系统处理客户请求的数量,直接体现软件系统的性能承载能力。

一般来说,吞吐量用请求数/秒或页面数/秒来衡量,从业务的角度,吞吐量也可以用访问人数/天或处理的业务数/小时等单位来衡量。从网络的角度来说,也可以用字节数/天等单位来考察网络流量。

4. 交易成功率

成功的交易数占总交易请求数的比率。

5. 资源利用率

资源利用率指的是服务器端的系统资源使用程度,主要有 CPU 利用率、内存利用率、磁盘 I/O 利用率、网络带宽利用率等。

17.1.2　性能测试的步骤

因为性能测试目的的特殊性,性能测试一般都要求系统要相对稳定。因此性能测试的介入时机一般是在一个项目的开发阶段的后期,目标系统达到一种相对稳定的状态之后进行。性能测试的实施一般遵循以下步骤:

1. 测试需求分析

测试需求分析主要通过与客户沟通和对系统的考察,确定客户对系统的真实性能需

求。全面了解目标系统的架构，因为不同架构的软件系统所适用的测试策略也会略有差别。另外调查分析系统的用户特点也是很重要的，比如系统使用高峰出现在工作日的哪个时段，总用户规模有多少等信息。可以归纳为以下几类内容：

- 用户数量；
- 业务功能使用特点；
- 用户分布（即使用不同功能的用户数）；
- 硬件配置环境（包括网络环境）；
- 软件配置环境；
- 当前系统数据量。

在此基础上可以进一步分析得出：

- 系统运行中所出现的问题有什么特征或规律，关键业务有哪些；
- 需要什么样的性能指标满足用户的使用要求等。

2．制定测试策略

根据上一步骤的性能测试需求结果，制定相应的测试策略，比如对于需求明确的性能符合性验证我们可以采用负载测试、疲劳强度测试进行验证；对于性能能力验证我们可以采用压力测试、疲劳测试、强度测试。

3．制定测试方案（计划）

性能测试方案一般包含测试需求、测试策略、测试场景、测试环境、人员及时间安排和风险分析等内容。其中测试需求和测试策略可以引用前两个步骤的成果，测试环境、人员及时间安排和风险分析与测试计划的制定没有多少差别，唯有测试场景是性能测试方案需要重点考虑的内容。

测试场景是系统生产环境下用户使用情景的模拟，测试场景包含以下内容：

- 业务描述（业务描述可以详细到操作步骤，便于脚本的录制或者编写）；
- 场景设置，如多少个用户，加压方式、用户退出方式等；
- 考察事务；
- 思考时间设置；
- 循环方式设置；
- 运行时间设置；
- 集合点设置；
- 监视的性能计数器设置。

4．开发测试脚本

依据测试场景的业务描述开发测试脚本。测试脚本可以手工编写，也可以使用专业的性能测试工具录制完成，或者两者结合起来使用。测试脚本开发完毕，需要进一步强化（如将固定值参数化，将动态数据进行关联，加入集合点、检查点等），以达到模拟实际使用状态的要求。

5．执行测试方案

根据测试方案，部署测试场景，包括每个场景的测试脚本（即测试功能），及每个测试

脚本对应的用户数、场景运行参数设置等;配置其他性能指标监控系统,比如服务器的资源利用率、网络监控等。

检查确认上述准备活动完成后,开始执行测试场景并监视其运行情况。场景运行完毕,保存测试结果,以备分析使用。

6. 分析测试结果

测试结果的分析实质上是对各类指标数据的分析过程,通过对测试结果数据的分析,查看测试结果是否满足要求,比如响应时间、资源利用率、吞吐量等。

7. 编写测试报告

根据测试结果编制系统的性能测试报告,对系统的性能状况给出评价和优化建议。

以上我们简明扼要地介绍了性能测试的基本概念、部分性能指标的介绍和性能测试的实施过程,但是在进行性能测试时还应注意到任何的性能考查都不是一个孤立的过程,系统的性能总是受着各种各样的因素的影响,单纯看一个指标是没有意义的。比如用户数很高,但是此时的系统资源利用率长期维持在一个很高的水平,那么这个用户数就是虚高。所以,我们在进行性能测试时,同时需要采集和监控系统应用环境的软硬件数据。这些指标数据包括:内存、处理器(CPU)和磁盘 I/O、网络状况等。数据库系统还要监控全局共享区大小的变化、数据库连接数和数据库死锁等。

当前,市面上可选的性能测试工具比较多,基本上都提供了"录制—回放"的脚本开发模式,易上手、易操作、易学习。本章主要向大家介绍 HP-Mercury 的 LoadRunner 和一款开源的性能测试软件 OpenSTA。

17.2 HP LoadRunner

17.2.1 概述

LoadRunner 是 HP 公司的一款性能测试工具,也是目前应用最为广泛的性能测试工具之一。应用 LoadRunner 可以模拟多用户实施并发负载,来测试软件系统的操作响应时间、点击数、吞吐量等,可以用来预测系统运行时的性能状况,如支持的最大并发用户数等,以及可以协助解决软件系统运行中遇到的性能问题,如系统运行慢、系统失灵等。

LoadRunner 具有如下功能特点:

① LoadRunner 可以记录下用户操作业务的流程,可以创建虚拟用户来模拟记录下的业务流程和真正用户的操作行为。

② LoadRunner 可以提供较为全面的测试数据,如操作系统、数据库、中间件等,可以让您在测试中,对各系统组件的运行性能进行评估,从而很快发现性能问题。

③ LoadRunner 支持多种通信协议,可以适用于多种体系架构的软件系统。

④ LoadRunner 有良好的操作界面和详细的帮助文档,可以使您快速掌握基本的操作流程。

LoadRunner 由四部分组成：

① VU Generator

用来生成测试脚本。VU Generator 捕获客户端与服务器端的信息交互，并生成测试脚本，同时提供相应的功能来对测试脚本进行增强，如脚本参数化等。

② Controller

用来模拟、部署和控制测试场景。通过 Controller 可以控制多个虚拟用户执行不同的测试脚本，同时可以监控操作系统、数据库、中间件等的资源利用情况，生成测试原始数据。

③ Analysis

用来分析测试结果。将得到的测试原始数据进行整理，以图表的形式给出各种测试统计数据，如不同时间各功能操作的响应时间、点击数、吞吐量、CPU 等的资源利用情况等。

④ Agent

用来生成虚拟用户。Controller 模拟的多个用户，可以由 Agent 来产生，Agent 可以跟 Controller 安装在一台机器上，也可以安装在不同的机器上。Controller 控制一个或多个 Agent 来产生多个虚拟用户执行测试脚本。

17.2.2 LoadRunner 的应用

1. 使用概述

1) VUG(虚拟用户生成器)

(1) 启动 VUG。VUG(Virtual User Generator，虚拟用户生成器，英文标识简称 VUG)在开始菜单的 Mercury LoadRunner 中的 Applications 目录，单击 Virtual User Generator 即可启动 VUG。

(2) VU Generator 主窗口介绍。VUG 主窗口主要由以下部分构成(如图 17-2 和图 17-3 所示)。

• Title Bar：显示当前脚本的名称；

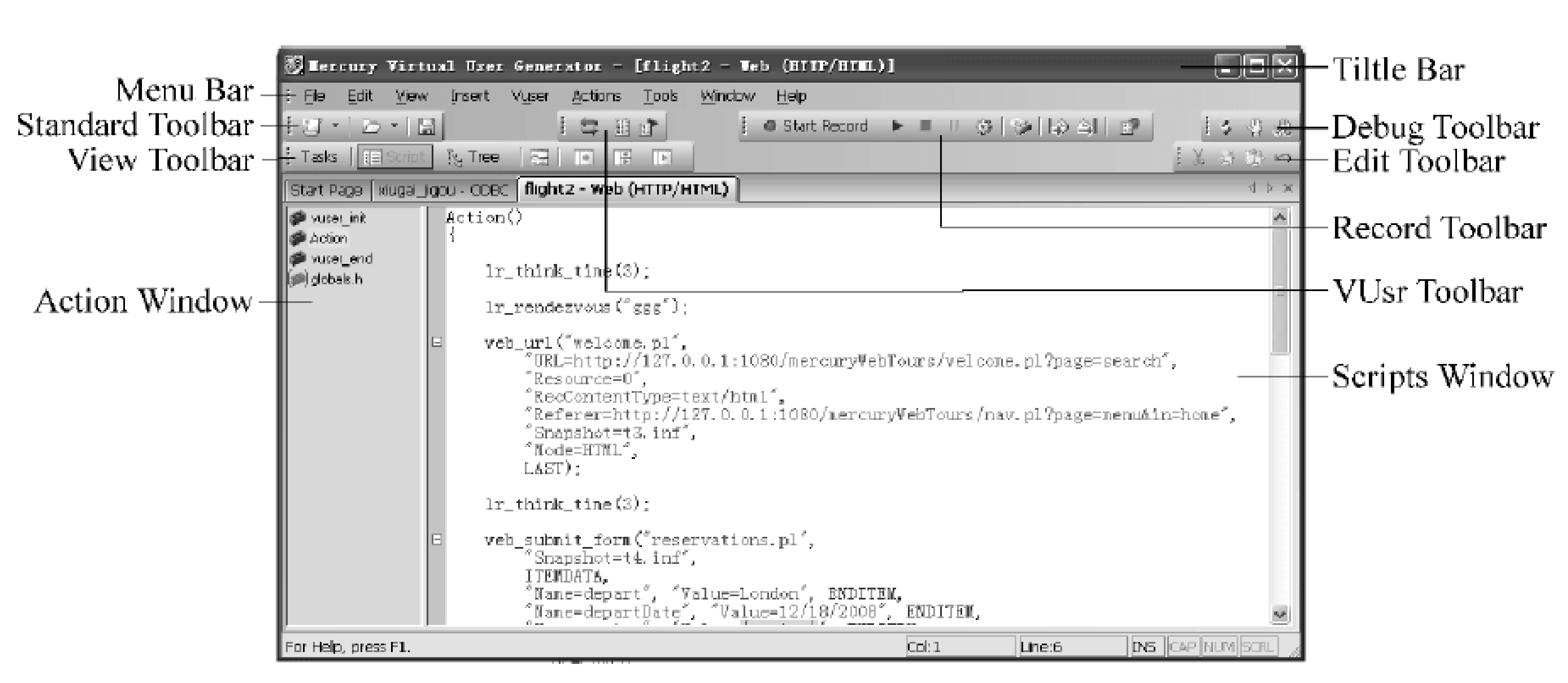


图 17-2　VUG 主窗口(Script 脚本显示方式)

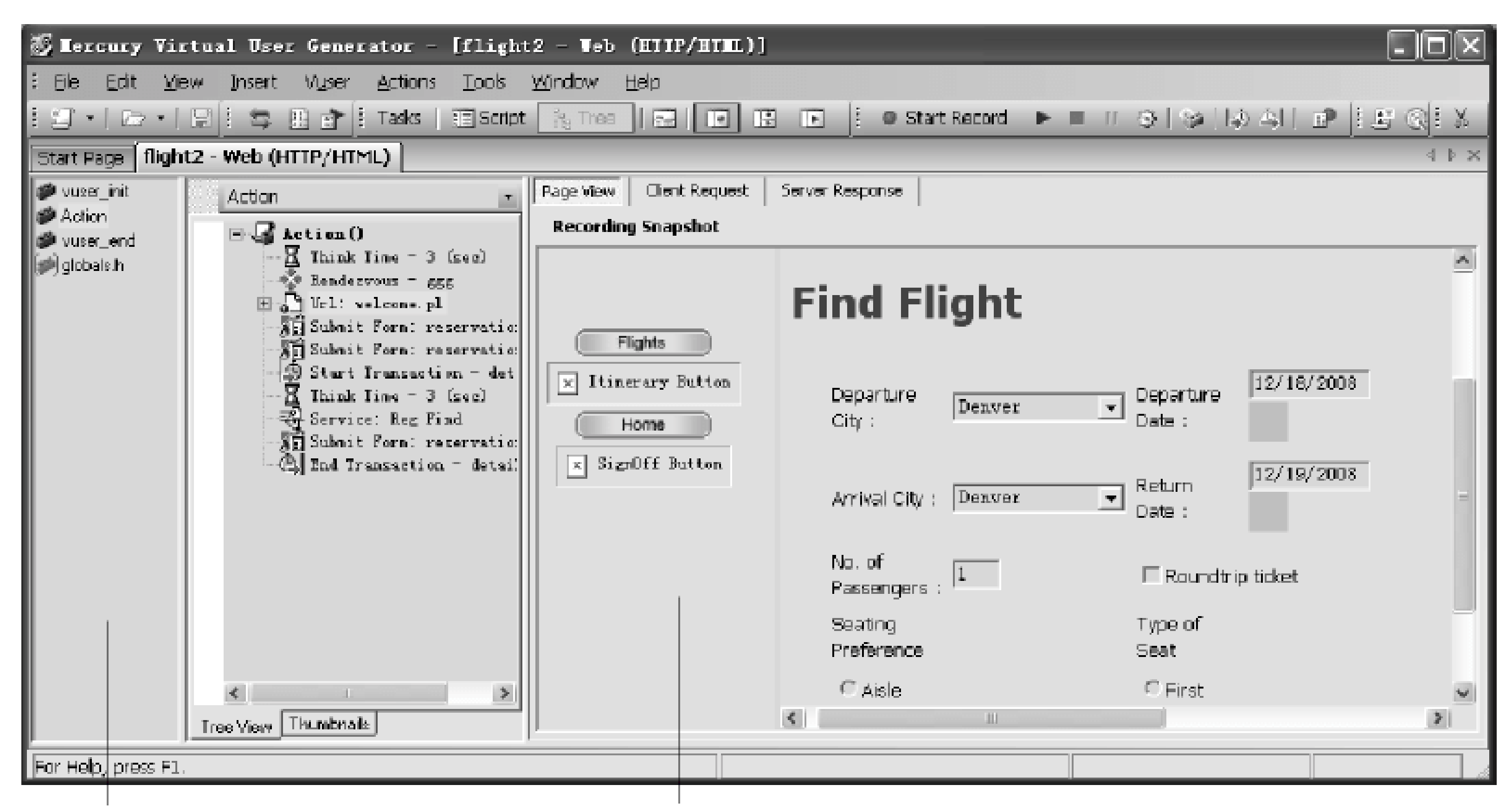


图 17-3　VUG 主窗口(Tree 脚本显示方式)

- Menu Bar：显示可用的菜单命令；
- Standard Toolbar：显示标准的操作命令；
- Debug Toolbar：显示调试脚本用的命令；
- View Toolbar：显示视图有关操作命令；
- Record Toolbar：显示录制脚本用的命令；
- Edit Toolbar：显示编辑脚本用的命令；
- VUsr Toolbar：显示脚本运行时用的命令；
- Action Window：显示当前脚本中的事务；
- Script Window：显示当前脚本内容，有 Tree 和 Script 两种显示方式可以选择。

2) Controller(控制器)

(1) 启动 Controller。Controller，称为控制器，主要用于性能测试场景的规划，在 LoadRunner 安装菜单中的 Applications 目录选择并单击 Controller 即可启动控制器。

启动 Controller 之后，出现“新建场景”窗口(如图 17-4 所示)。

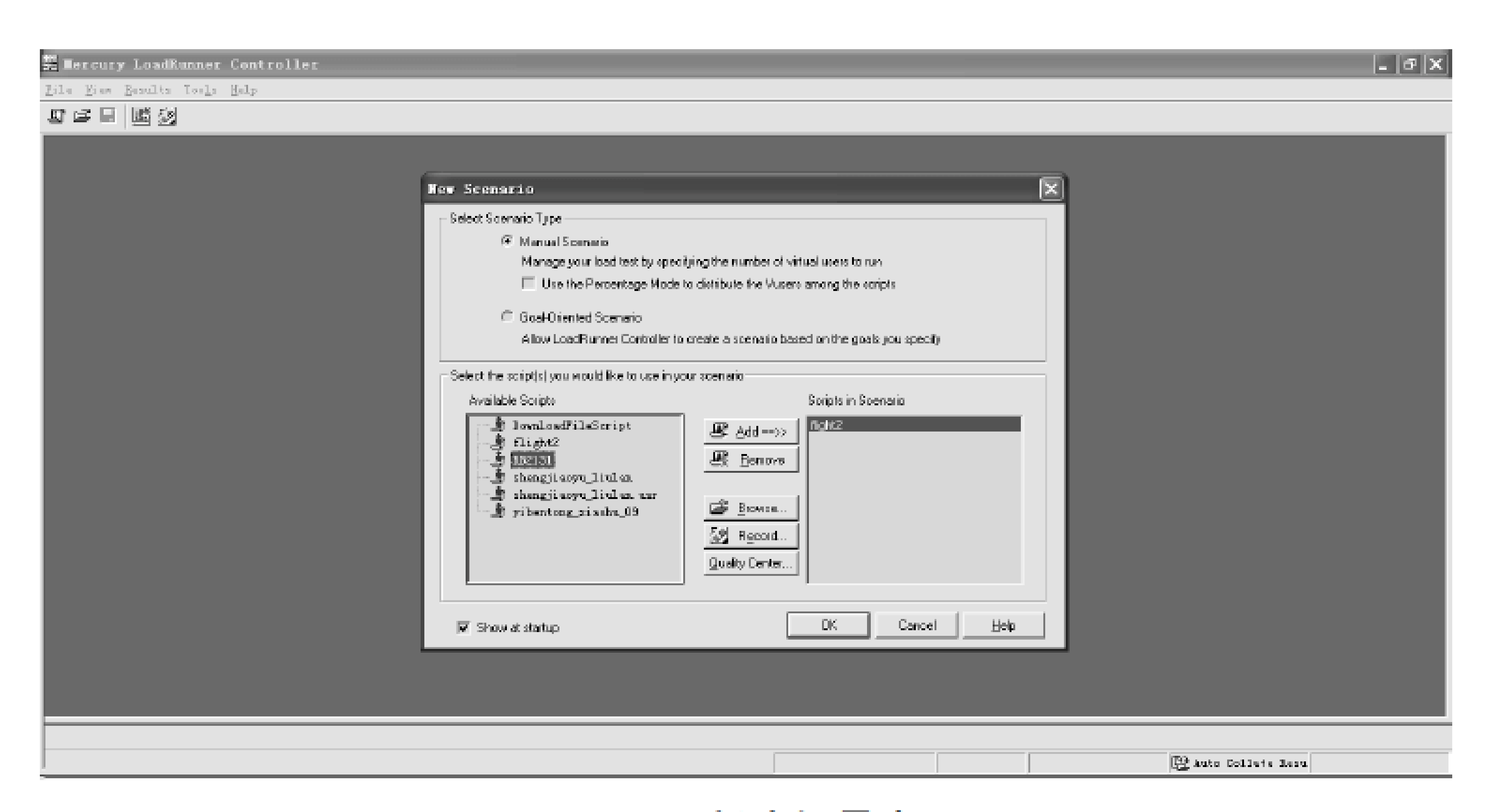


图 17-4　新建场景窗口

在这里，可以方便地根据实际测试需求设置场景模式，并将 VUG 中录制开发的业务测试脚本加入到测试场景中，单击 OK 按钮，即可进入 LoadRunner 的控制器(Controller)主窗口。

(2) Controller 主窗口。Controller 的主要窗口主要包括两大部分：Design Tab(设计规划栏)和 Run Tab(运行监控栏)。其中，Design Tab(设计规划栏)主要用于场景运行时设置，Run Tab(运行监控栏)主要用于场景运行和监视。

Design Tab 窗口主要由以下几部分构成(如图 17-5 所示)：

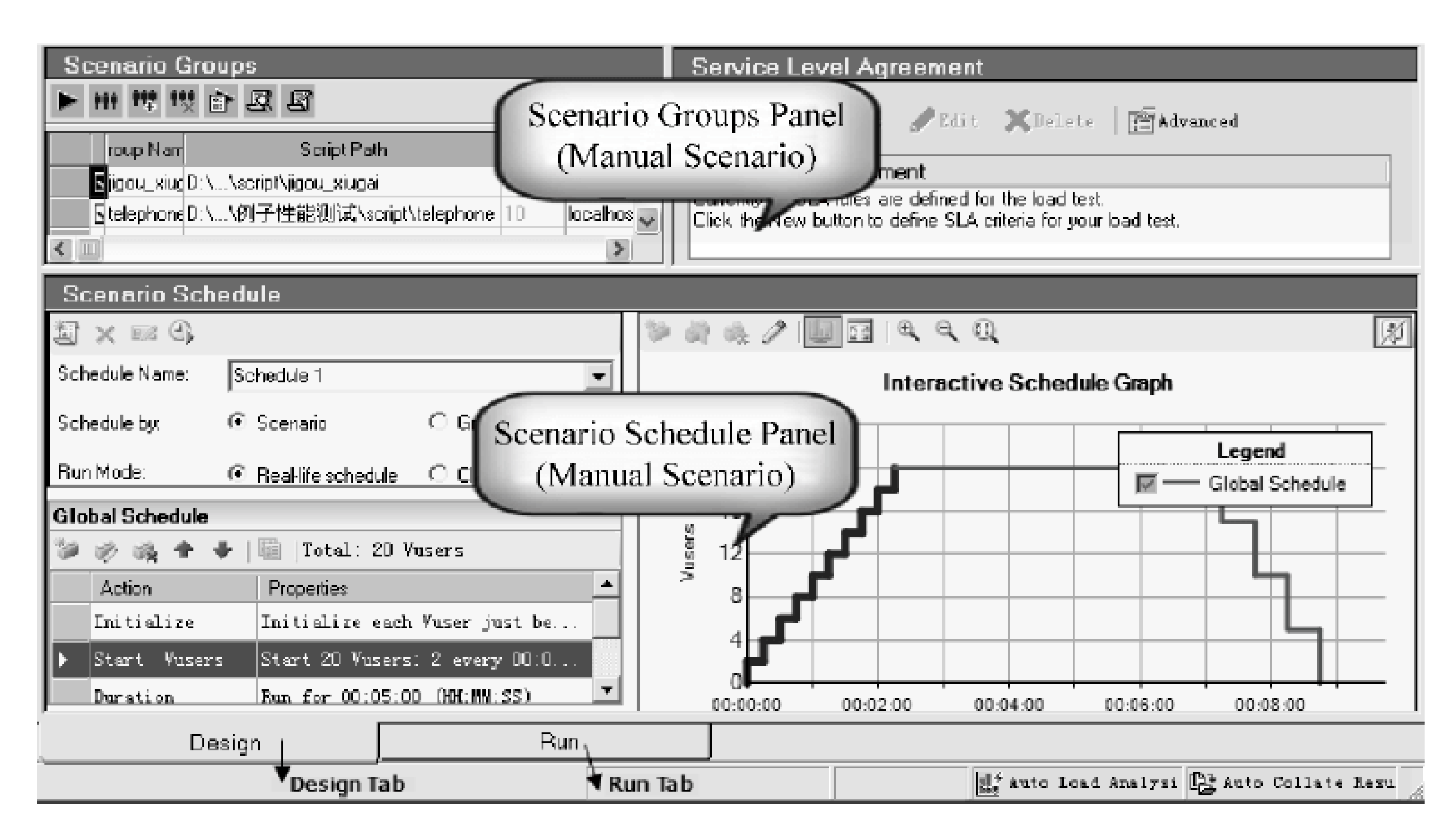


图 17-5　Design Tab 窗口构成

- Scenario Schedule panel(Manual Scenario)：显示当前场景运行设置情况；
- Scenario Groups panel(Manual Scenario)：显示当前场景包含的场景组(Group Name)，每个场景组包含的测试脚本(Script Path)，以及每个测试脚本对应的用户数(Quantity)和分配的虚拟用户生成器(Load Generators)、运行参数(Run Time Settings)等。

Run Tab 窗口主要由以下几部分构成(如图 17-6 所示)：

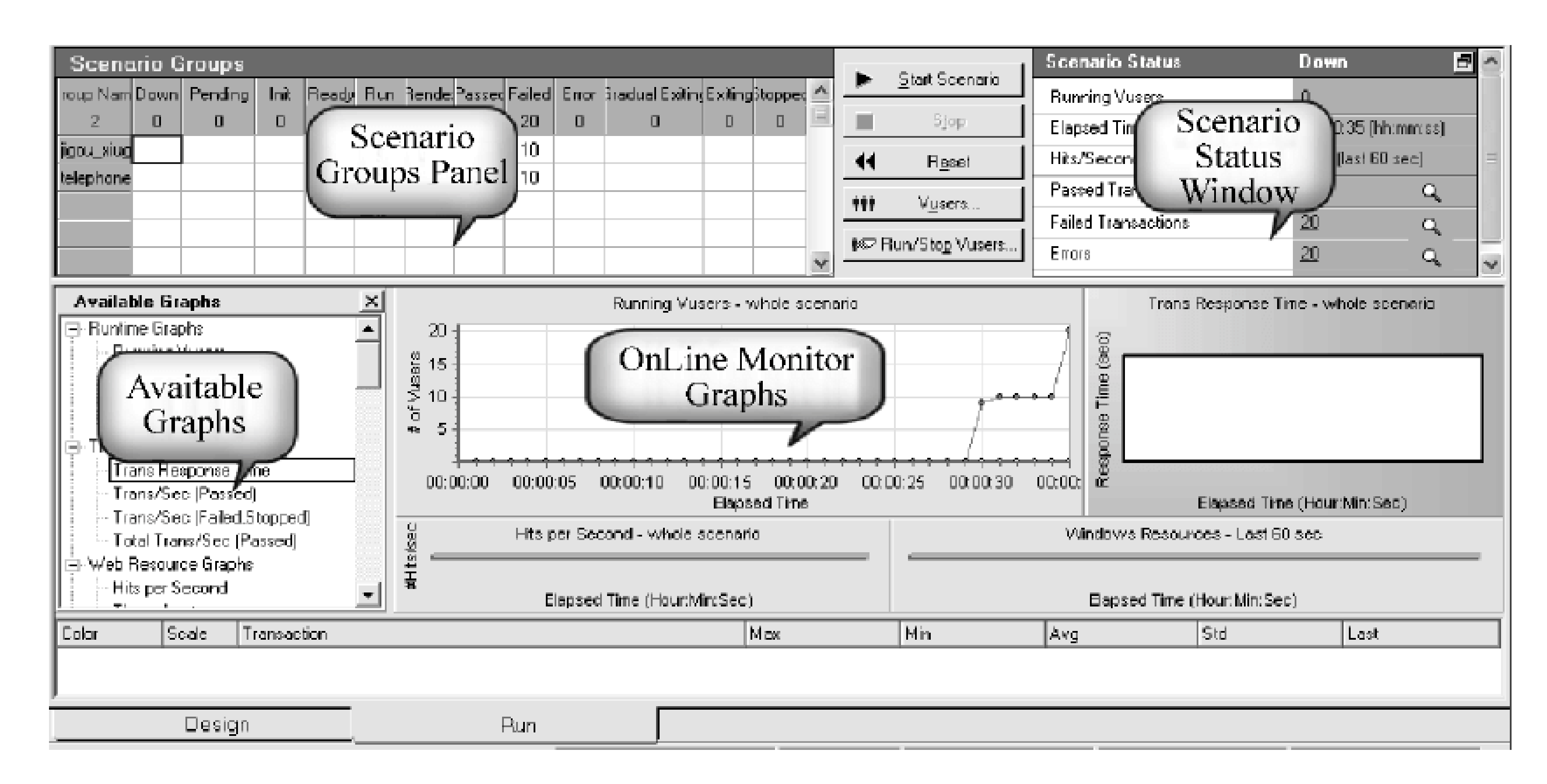


图 17-6　Run Tab 窗口构成

- Scenario Group panel：显示各场景组的当前运行状态；
- Scenario Status Window：显示场景的运行状态，包括当前运行的用户数、场景持续时间、每秒的点击数、通过的交易数、失败的交易数、出现的错误等；
- Avaitable Graphs：显示可用的监视图像类型；
- OnLine Monitor Graphs：显示监视数据的变化曲线。

3）Analysis(分析器)

(1) 启动 Analysis。Analysis，称为分析器，主要用于性能测试各类数据的采集和分析，要启动 Analysis 只需在 LoadRunner 安装菜单的 Applications 中选择并单击 Analysis 即可。启动 Analysis，还有另外一种方式，即在 Controller 中开启 Auto Load Analysis(如图 17-7 所示)。

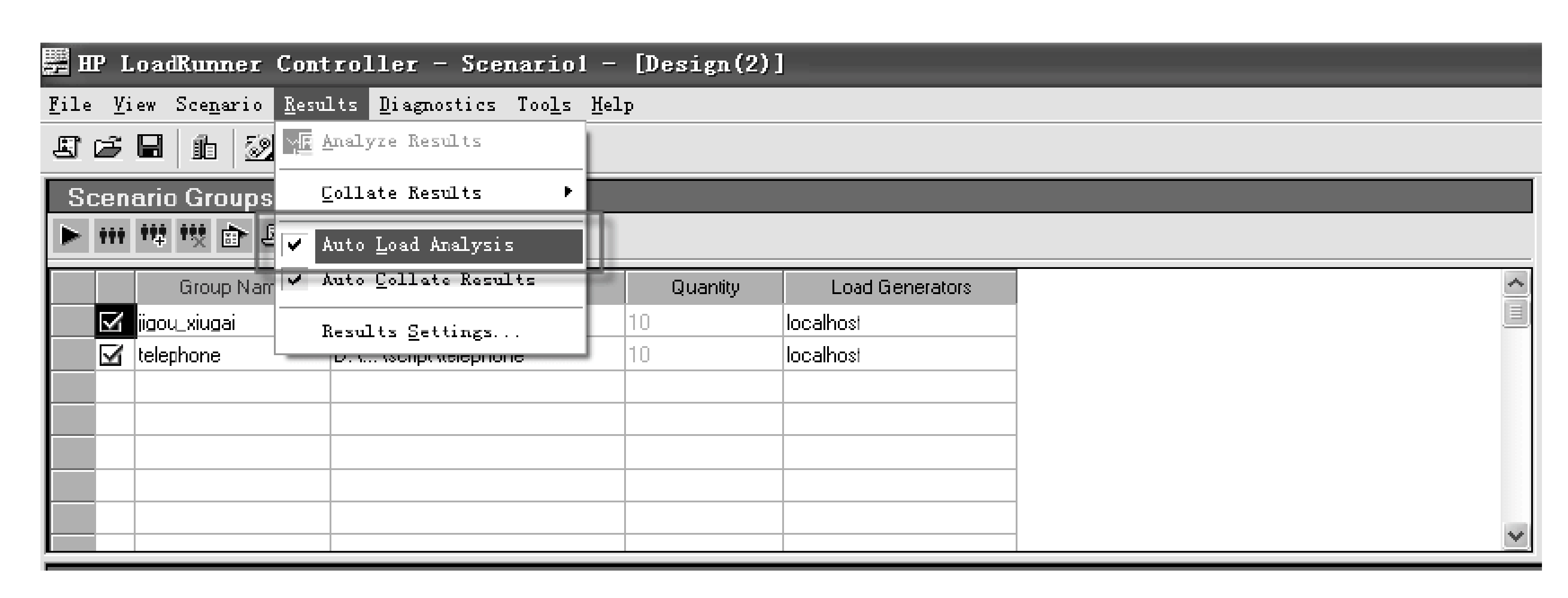


图 17-7　Auto Load Analysis

(2) Analysis 窗口主要构成部分如图 17-8 所示。

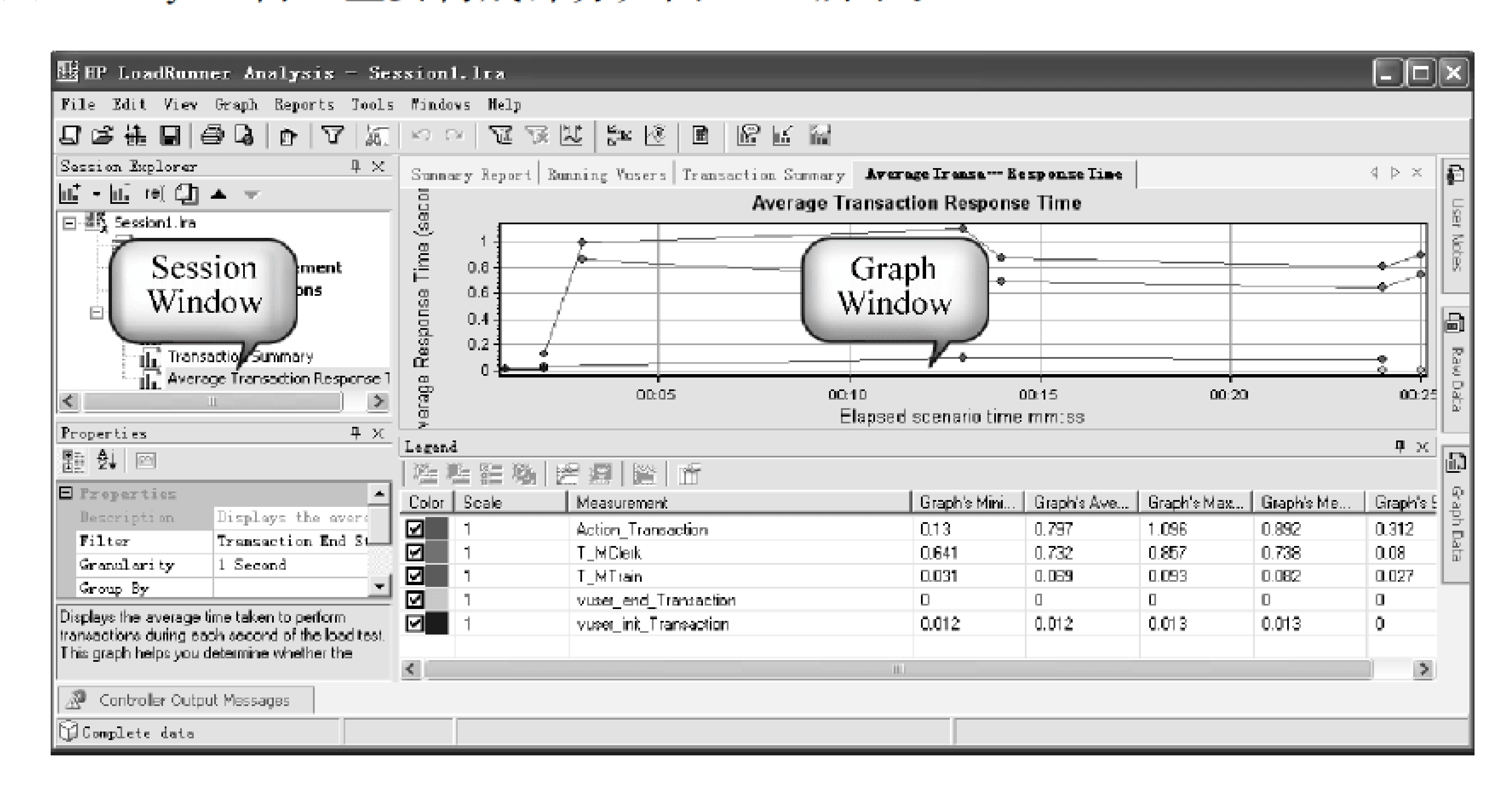


图 17-8　Analysis 窗口构成

- Session Window：显示测试结果的数据、图形类别；
- Graph Window：以图形的方式显示具体的测试结果。

4）Agent(代理)

Agent 是 LoadRunner 的代理进程，一般 LoadRunner 的默认设置是 Agent 随计算机

的启动自动运行。Agent 正常启动后，会以系统图标的形式显示在 Windows 的任务栏上。

2. LoadRunner 测试流程

LoadRunner 的性能测试流程如下：

1）制定测试计划

测试计划是指导测试执行的依据，一个好的测试计划有利于 LoadRunner 顺利地完成测试目标。

2）准备测试脚本

根据指定的测试用例，准备相应的测试脚本，同时对测试脚本进行增强，如参数化等。

3）创建测试场景

测试场景是模拟软件系统实际应用中的一些情况，包括测试脚本、每个测试脚本的用户数量、数据库容量、执行过程中的相关参数设定，如思考时间等。

4）运行测试场景

单击 LoadRunner 的场景运行按钮，运行创建的各种测试场景。在各场景运行过程中，监视各种测试数据，包括各测试功能的响应时间、各服务器、数据库、中间件的资源利用情况等。

5）分析测试结果

根据测试得到的数据结果，进行深入地分析。

3. LoadRunner 应用举例

本节通过 LoadRunner 自带的飞机订票的例子来讲述 LoadRunner 的使用。

1）录制测试脚本

启动 LoadRunner 的虚拟用户生成器，并根据测试的软件选择协议，这里选择 Web（HTTP/HTML）协议（如图 17-9 所示）。

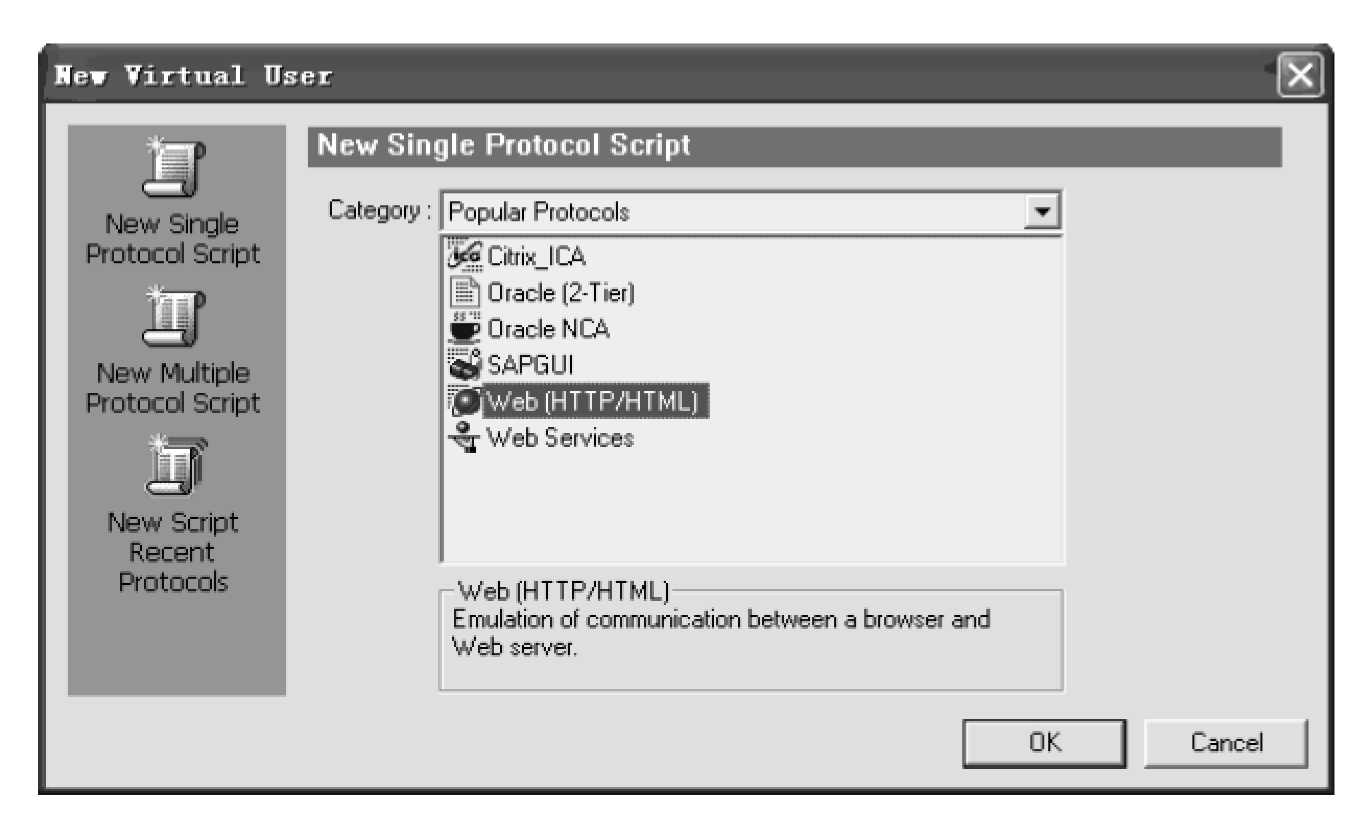


图 17-9　协议选择

在地址栏中输入飞机订票系统的地址“http://127.0.0.1:1080/mercuryWebTours/”，进行飞机订票的业务功能操作，录制性能测试脚本。

录制过程中，可以根据需要自定义事务。这里我们对 Payment Details 信息处理单独定义事务，以便单独考查该步操作的性能状况。如图 17-10 所示，单击录制工具条中的 Insert Start Transaction 图标，并输入事务名称：



图 17-10　插入事务

订票操作完毕，单击停止录制按钮，结束测试脚本的录制，得到订票操作的完整测试脚本，并显示在 LoadRunner 脚本编辑器中（如图 17-11 所示）。

```
Action()
{

    lr_think_time(15);

    web_url("welcome.pl",
        "URL=http://127.0.0.1:1080/mercuryWebTours/welcome.pl?page=search",
        "Resource=0",
        "RecContentType=text/html",
        "Referer=http://127.0.0.1:1080/mercuryWebTours/nav.pl?page=menu&in=home",
        "Snapshot=t3.inf",
        "Mode=HTML",
        LAST);

    lr_think_time(12);

    web_submit_form("reservations.pl",
        "Snapshot=t4.inf",
        ITEMDATA,
        "Name=depart", "Value=London", ENDITEM,
        "Name=departDate", "Value=12/18/2008", ENDITEM,
        "Name=arrive", "Value=Denver", ENDITEM,
        "Name=returnDate", "Value=12/19/2008", ENDITEM,
        "Name=numPassengers", "Value=1", ENDITEM,
        "Name=roundtrip", "Value=<OFF>", ENDITEM,
        "Name=seatPref", "Value=None", ENDITEM,
        "Name=seatType", "Value=First", ENDITEM,
        "Name=findFlights.x", "Value=48", ENDITEM,
        "Name=findFlights.y", "Value=9", ENDITEM,
        LAST);

    lr_think_time(6);

    web_submit_form("reservations.pl_2",
        "Snapshot=t5.inf",
        ITEMDATA,
        "Name=outboundFlight", "Value=200;635;12/18/2008", ENDITEM,
        "Name=reserveFlights.x", "Value=46", ENDITEM,
        "Name=reserveFlights.y", "Value=12", ENDITEM,
        LAST);
```

图 17-11　订票操作的脚本

2）增强测试脚本

根据测试需要，需要增强测试脚本，如进行参数化、定义集合点等。这里选择“目的地”进行参数化，通过阅读脚本，找到“目的地”字段的值，单击鼠标右键，在快捷菜单中选择 Replace with a parameter（如图 17-12 所示）。

接下来选取默认的参数类型，即文件类型，新建参数文件，并输入参数具体的取值集合（如图 17-13 所示）。

测试脚本参数化工作结束后，单击工具栏中的“运行”，或者通过菜单中的 Run 进行脚本回放，验证测试脚本的正确性。这个过程称之为“脚本调试”。

3）创建测试场景

测试脚本准备完毕，开始创建测试场景。首先启动 LoadRunner 的 Controller，选择要运行的测试脚本（如图 17-5 所示）。然后对各测试脚本进行相应的设置，例如分配的用户数、加压方式和脚本运行时设置等，以及对测试场景进行相应的设置等。选中测试脚本，单击 Controller 界面中的 Run-Time Settings 进行相应设置，这里我们设置测试脚本

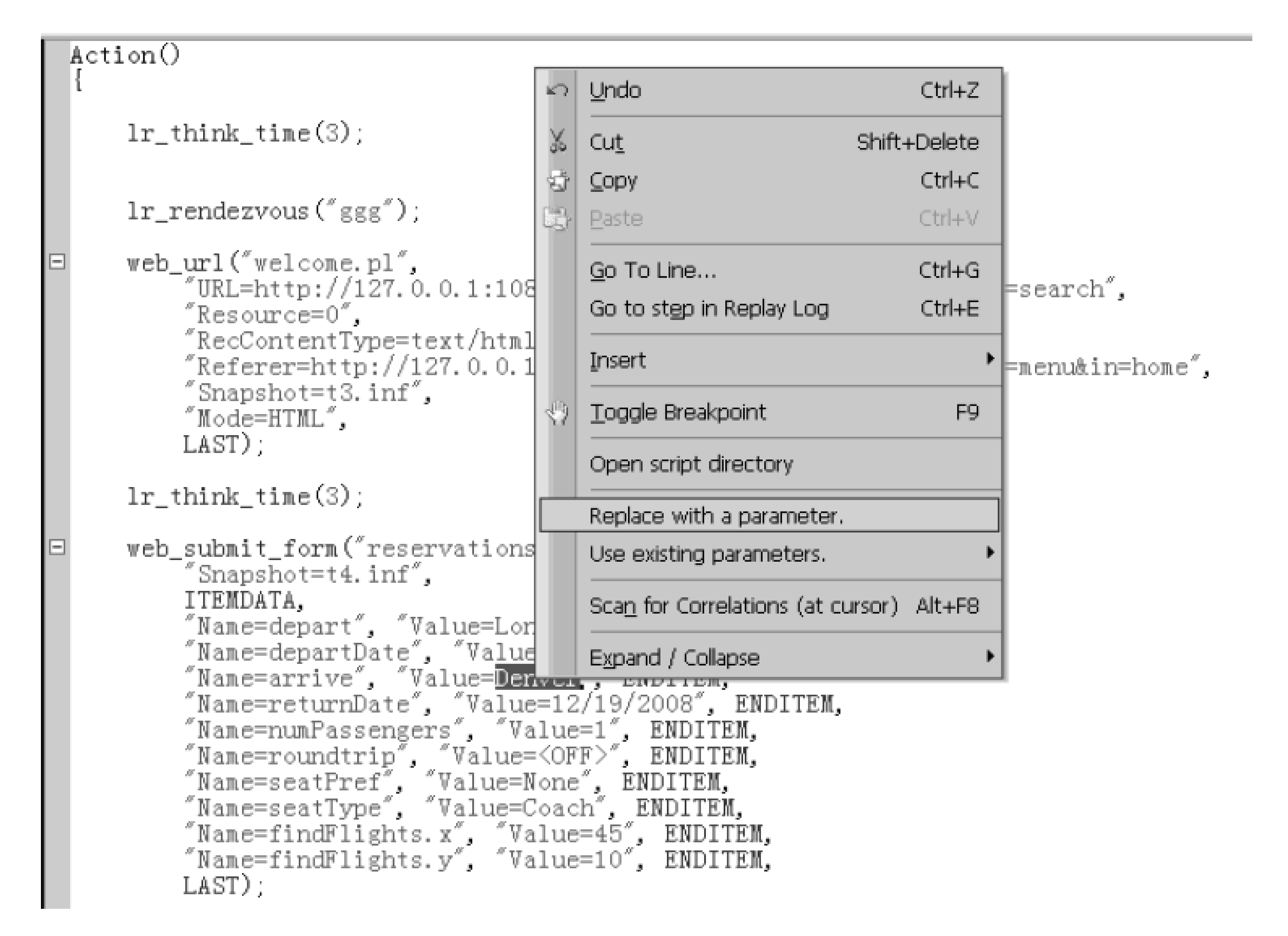


图 17-12　参数化示例

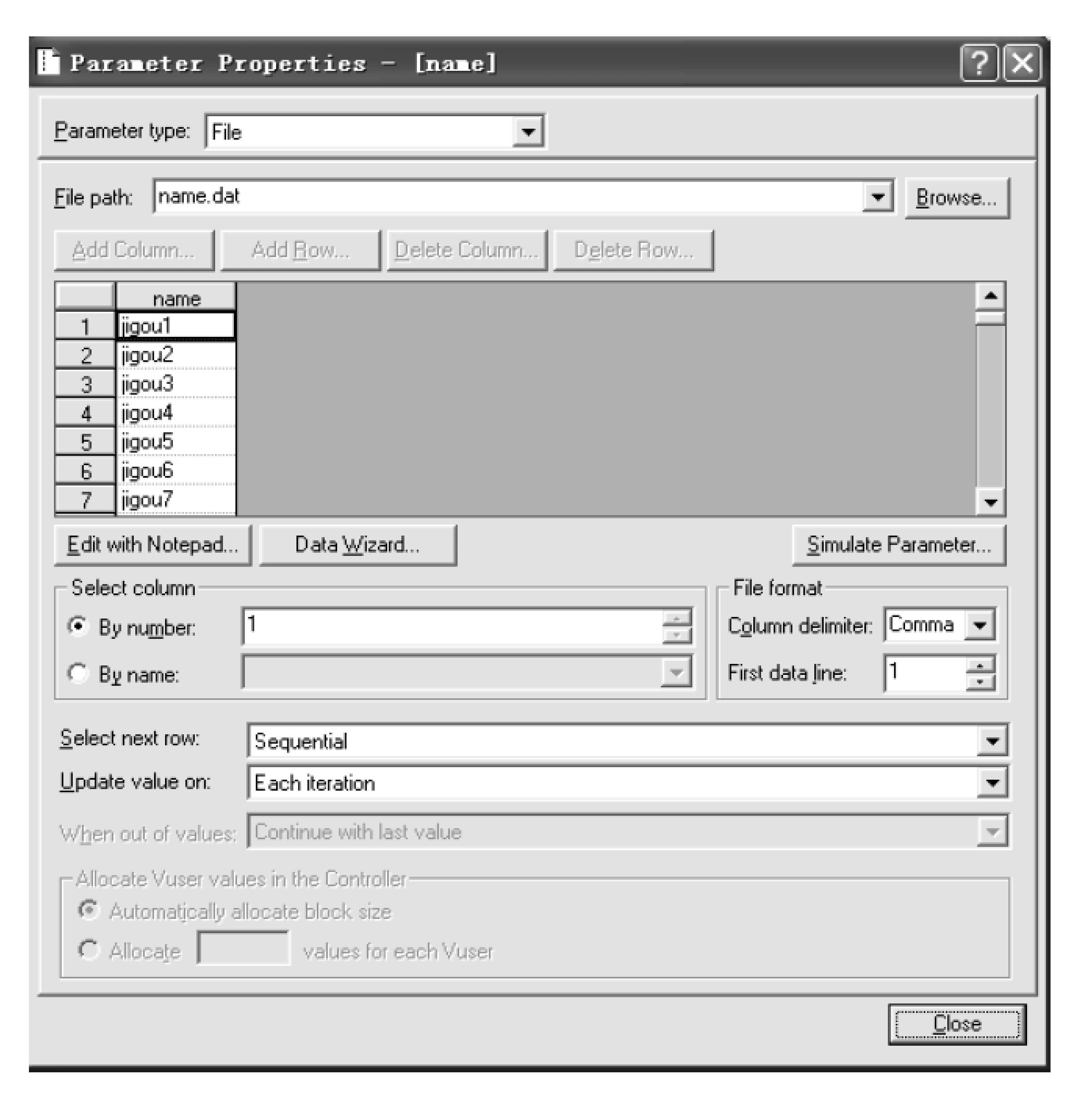


图 17-13　参数属性设置

中 Action 部分连续运行 3 次(如图 17-14 所示)。

4) 运行测试场景

测试场景部署完毕,开始运行测试场景,单击 Controller 中的运行按钮,开始运行测

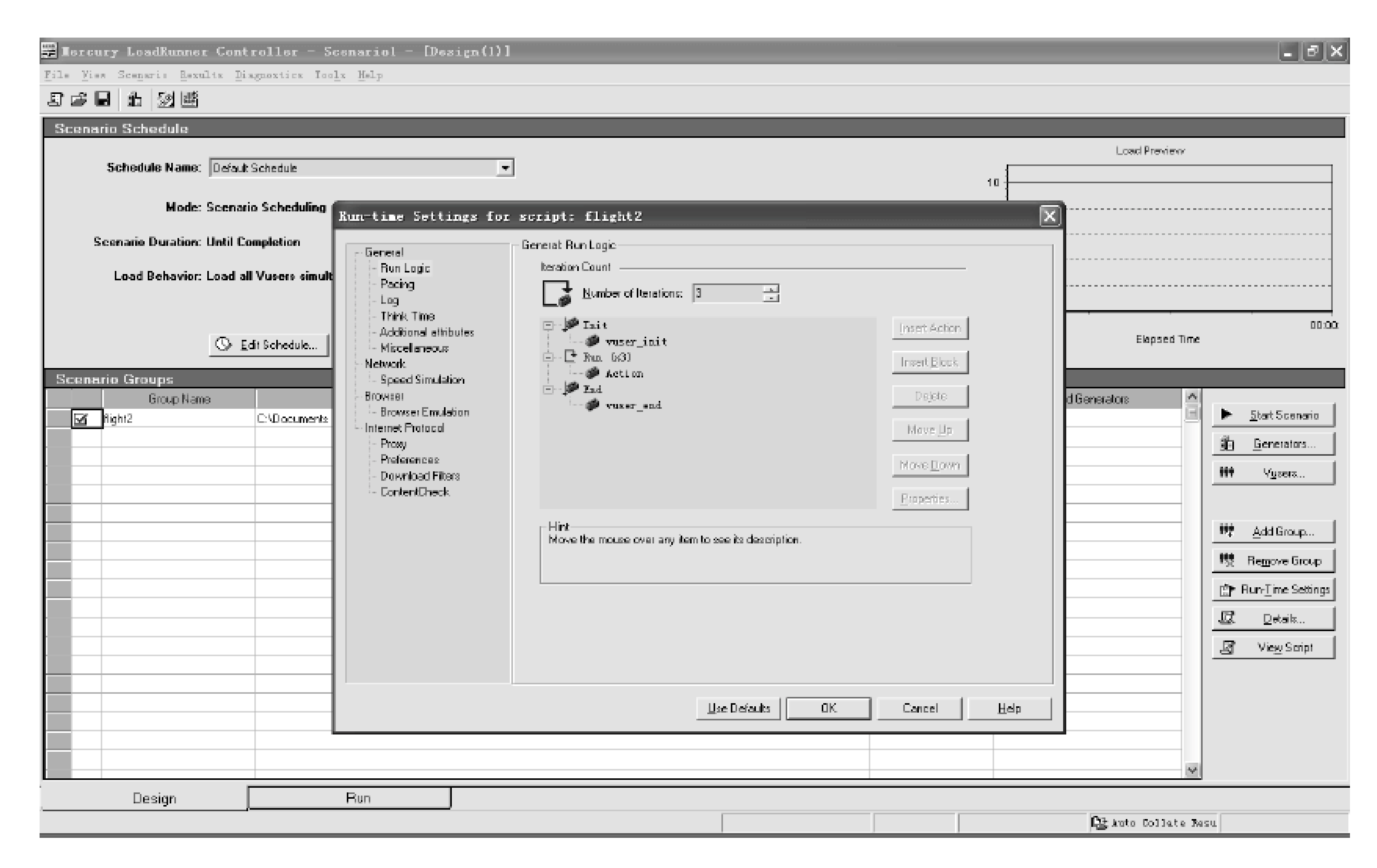


图 17-14　场景设置

试场景，场景运行的同时，监视场景的运行情况，包括虚拟用户的加载情况、各测试功能的响应时间情况、服务器资源利用率情况等（如图 17-15 所示）。

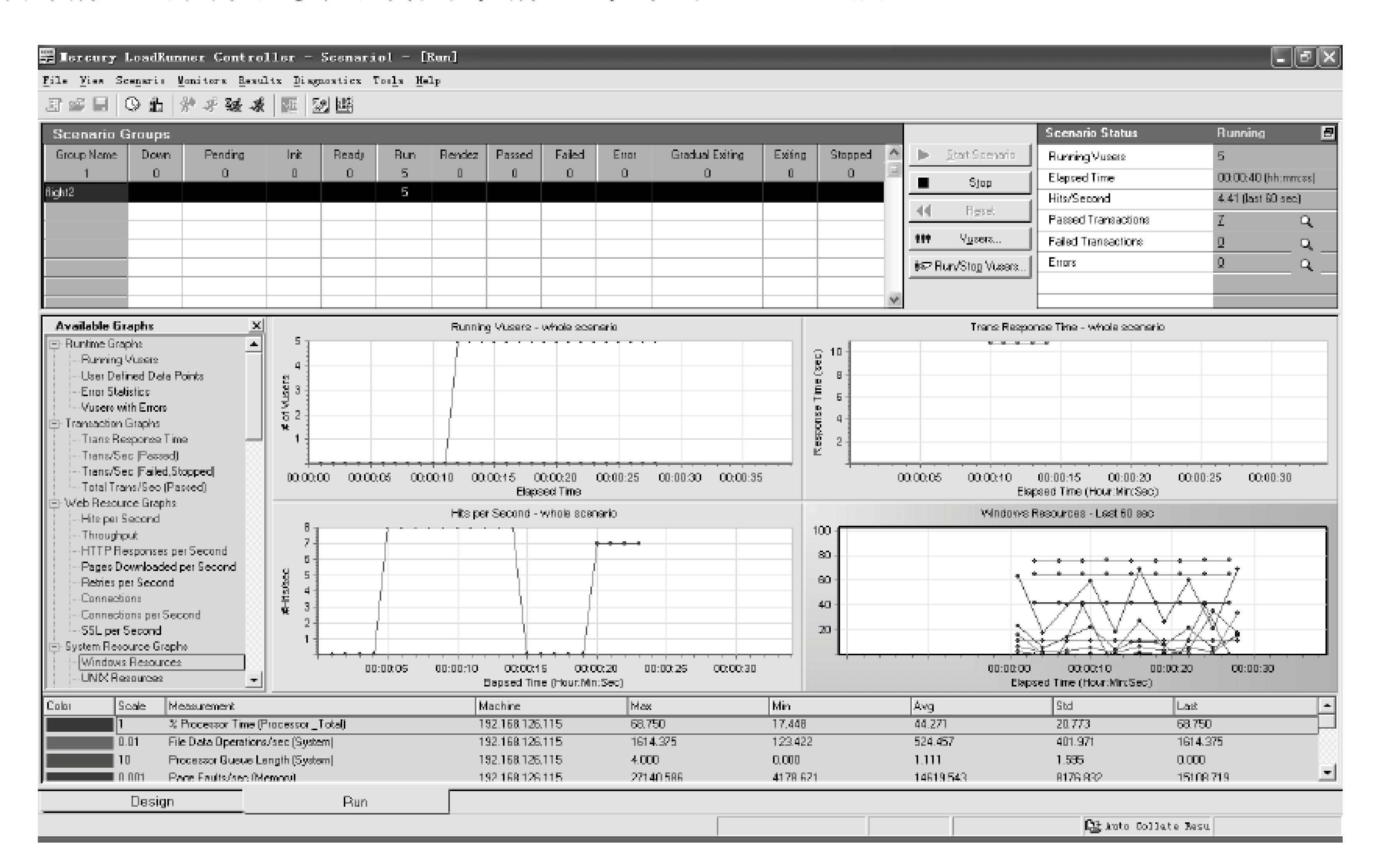


图 17-15　场景运行监控

5）分析测试结果

场景运行完毕，单击 Controller 界面菜单中的 Results/Analyze Results，LoadRunner 会自动收集分析测试结果，并给出图文结合的各类统计分析报告，例如运行通过的事务、运行失败的事务、各功能的响应时间、吞吐量、服务器的资源利用等曲线图（如图 17-16 所示）。根据 LoadRunner 提供的测试数据，进行综合分析，从而达到对系统的性能状况进

行评估和预测的目的。

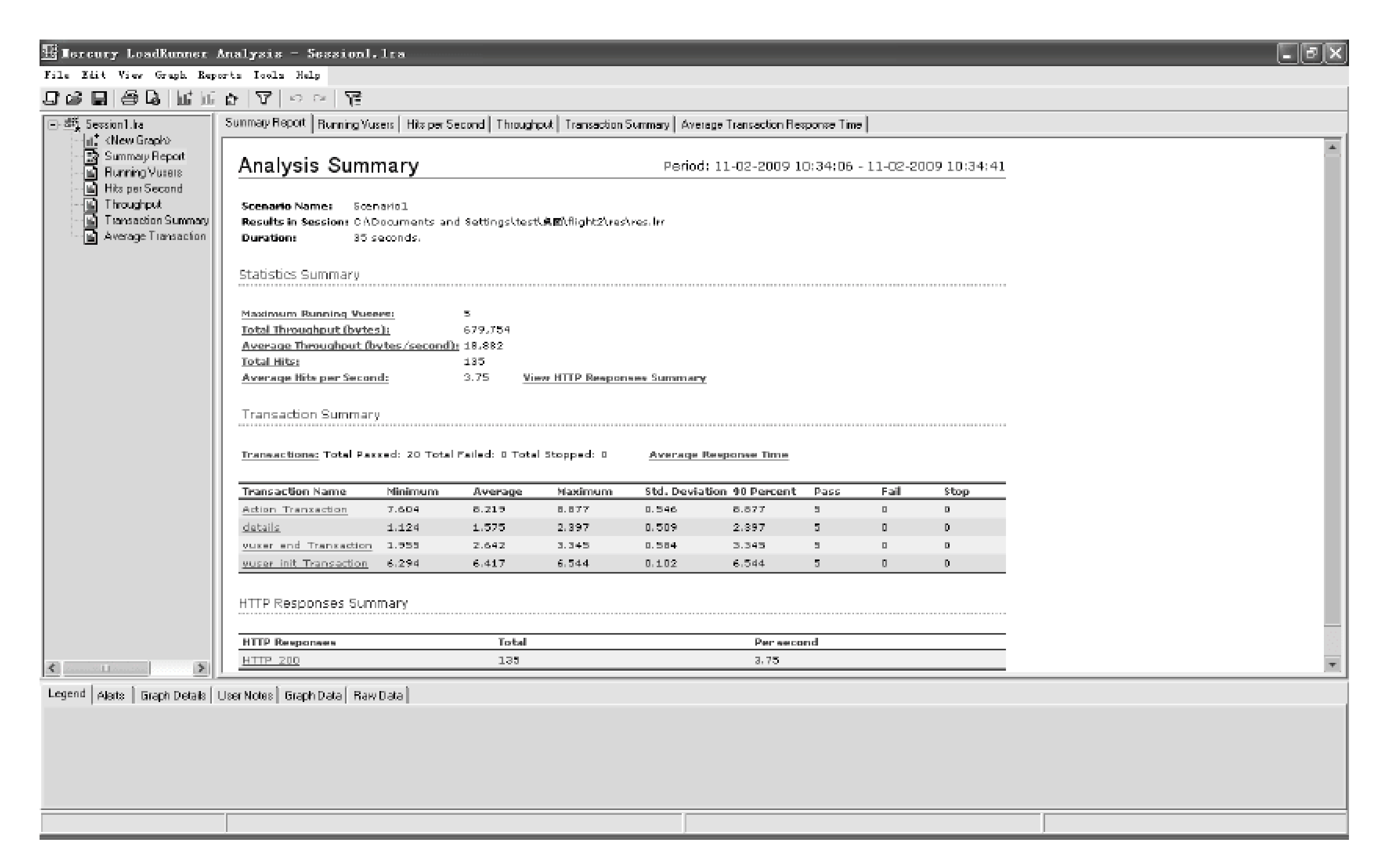


图 17-16　测试结果分析

17.2.3　常见问题解答

1. 协议选择

根据被测系统的通信协议来确定录制脚本时应选取的协议。一般对于常见的应用软件,可以根据软件的结构来选择协议:

B/S 结构,选择 Web(HTTP/HTML)协议;C/S 结构,可以根据后端数据库的类型来选择,如 SybaseCTLib 协议用于测试后台的数据库为 Sybase 的应用;MS SQL Server 协议用于测试后台数据库为 SQL Server 的应用。对于一些没有数据库的 Windows 应用,可选用 Windows Sockets 协议,一般情况下都可以成功录制到脚本,因为几乎所有的网络传输中都是基于 TCP 协议或 UDP 协议的,而 Windows Sockets 则是底层的协议。

2. HTML-based 与 URL-based 模式的选取问题

HTML-based 方式对每个页面录制形成一条语句,对 LoadRunner 来说,在该模式下,访问一个页面,首先会与服务器之间建立一个连接获取页面的内容,然后从页面中分解得到其他的元素(component),然后建立几个连接分别获取相应的元素。

URL-based 方式将每条客户端发出的请求录制成一条语句,对 LoadRunner 来说,在该模式下,一条语句只建立一个到服务器的连接,LoadRunner 提供了 Web_concurrent_start 和 Web_concurrent_end 函数模拟 HTML-based 的工作方式。

对于录制方式的选择,可以遵循如下原则:

- 如果应用是 Web 应用,首选是 HTML-based 方式;

- 如果应用是使用 HTTP 协议的非 Web 应用，首选是 URL-based 方式；
- 如果 Web 应用中使用了 Java applet 程序，且 applet 程序与服务器之间存在通信，选用 URL-based 方式；
- 如果 Web 应用中使用了 Javascript、vbscript 脚本与服务器之间存在通信(调用了服务端组件)，选用 URL-based 方式。

3. 参数选取设置

当脚本参数类型为 Data File 形式时，脚本运行参数更新方式设置选项主要有 Select next row、Update value on、When out of values(如图 17-17 所示)。

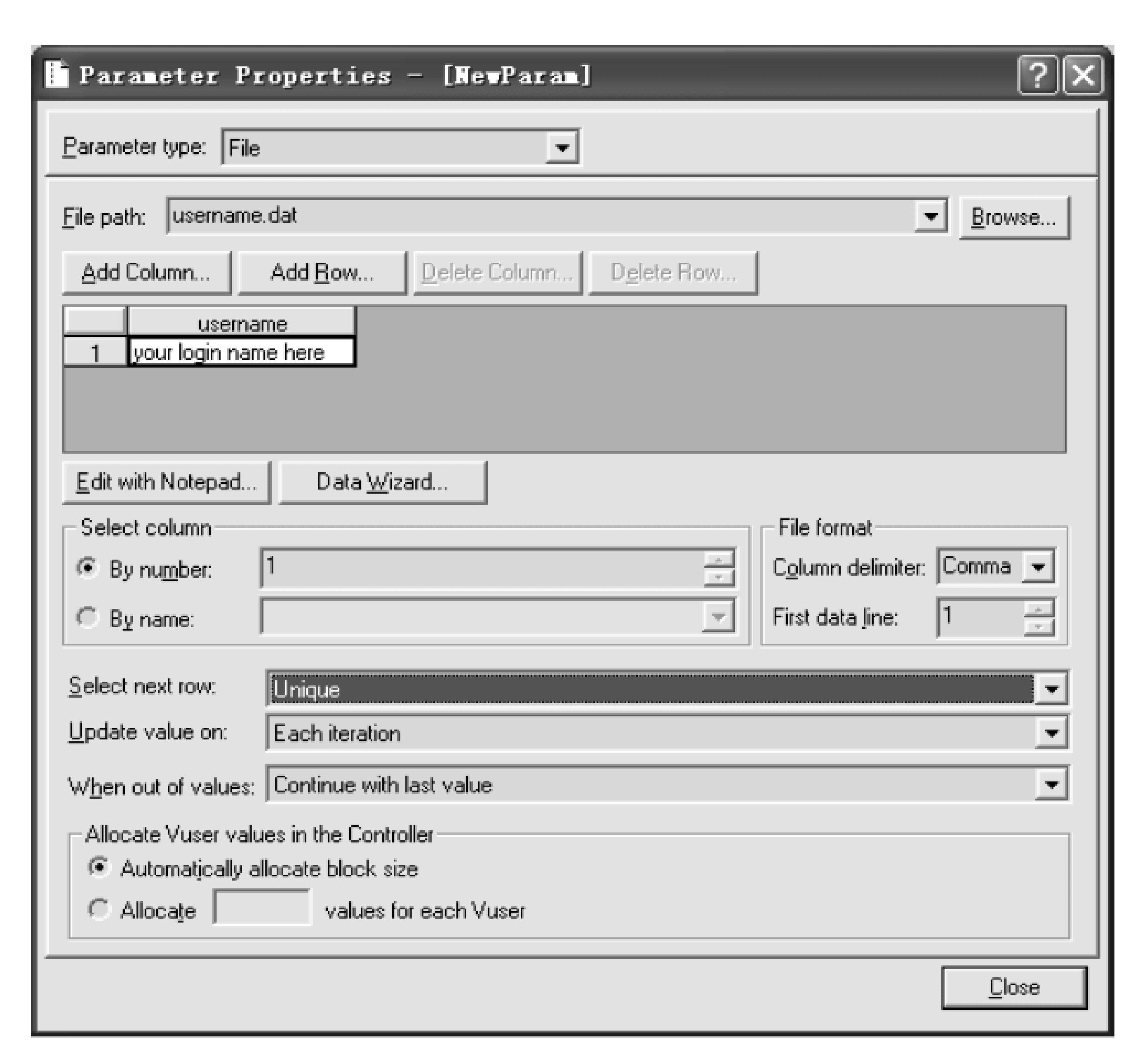


图 17-17 参数属性设置

Select next row 指明以何种方式从数据池中获取数据。这个选项共有 3 种方式，分别是 Sequential、Random、Unique，其中 Sequential 表示"按照顺序取值"，Random 表示"从数据池中随机取值"，Unique 表示"每次取唯一值"。

Update value on 指明参数值何时发生改变。这个选项也有 3 种方式：Each Iteration、Each Occurrence、Once，其中 Each Iteration 表示"在每次迭代时更新参数的值"，Each Occurrence 表示"在参数每次出现时更新参数的值"，Once 表示"只在第一次迭代时为参数取一次值，以后每次迭代都使用该值"。

When out of values 只在 Select next row 设置为 Unique 时才有效，指明当参数值取完之后如何处理。这个选项 LoadRunner 提供了 3 种处理方式：Abort VUser、Continue in a cyclic manner、Continue with last value。Abort VUser 表示停止运行虚拟用户，Continue in a cyclic manner 表示重新从数据池的第一个取值开始循环，Continue with

last value 表示之后的参数值都用最后一个取值来替代。

从上述规则可以看出，即便是对应同一个参数文件，对应 Select next row 和 Update value on 的不同组合，虚拟用户运行过程中为参数所取的值也是不同的。下面以图 17-18 所示参数文件为例，说明虚拟用户运行过程中是如何为参数匹配数据的。



图 17-18　参数文件

为了便于表述和理解，假定测试场景是这样的：场景中只存在一个测试脚本且测试脚本中有两个地方用到了参数 name，我们分配两个虚拟用户执行这个测试脚本，每个用户执行两遍，我们来看一下参数取值规则的不同组合方式所对应的参数取值结果（见表 17-1）。

表 17-1　不同参数取值规则的参数取值

Select next row	Update value on	参 数 取 值
Sequential	Each Iteration	两个虚拟用户的参数选取方式相同。脚本中两处的参数 name，在第一次迭代时取值都为 aaa，第二次迭代时取值都为 bbb
	Each Occurrence	两个虚拟用户的参数选取方式相同。第一次迭代时，脚本中第一处的参数 name 取值都为 aaa，第二处的参数 name 取值都为 bbb；第二次迭代时，脚本中第一处的参数 name 取值都为 ccc，第二处的参数 name 取值都为 ddd
	Once	两个虚拟用户的参数选取方式相同。所有的参数 name 取值都为 aaa
Random	Each Iteration	两个虚拟用的参数选取方式相同。第一次迭代时，两处的参数 name 随机取相同的值，如 ccc，第二次迭代时，两处的参数 name 随机取相同的值，如 bbb
	Each Occurrence	两个虚拟用的参数选取方式相同。第一次迭代时，第一处的参数 name 随机取值，如 ccc，第二处的参数 name 随机取值，如 aaa；第二次迭代时，第一处的参数 name 随机取值，如 bbb，第二处的参数 name 随机取值，如 aaa
	Once	两个虚拟用户的参数选取方式相同。所有的参数 name 随机取同一值如都为 ddd

续表

Select next row	Update value on	参 数 取 值
Unique	Each Iteration	两个虚拟用户的参数选取方式不同。第一个虚拟用户在第一次迭代时,两处的参数 name 选取相同的值,都为 aaa,第二次迭代时,两处的参数 name 选取相同的值,都为 bbb;第二个虚拟用户在第一次迭代时,两处的参数 name 选取相同的值,都为 ccc,第二次迭代时,两处的参数 name 选取相同的值,都为 ddd
	Each Occurrence	两个虚拟用户的参数选取方式不同。第一个虚拟用户在第一次迭代时,第一处的参数 name 取值 aaa,第二处的参数 name 取值 bbb,第二次迭代时,第一处的参数取值 ccc,第二处的参数 name 取值 ddd;第二个虚拟用户在第一次迭代时,第一处的参数 name 取值 eee,第二处的参数 name 取值 fff,第二次迭代时,第一处的参数取值 ggg,第二处的参数 name 取值 hhh
	Once	两个虚拟用户的参数选取方式不同。第一个虚拟用户两次迭代时,两处的参数 name 选取相同的值,都为 aaa;第二个虚拟用户两次迭代时,两处的参数 name 选取相同的值,都为 bbb

4. 字符乱码

在录制脚本中,如果遇到乱码的问题,可以尝试如下方法来解决:

① 在 vugen 菜单中,选择 Tools→Recoding Options→Advanced→Support charset 命令,选中 UTF-8。

② 把 IE 的编码选择成 UTF-8。

③ 使用 lr_convert_string_encoding 函数进行转换。

17.3 OpenSTA

17.3.1 概述

OpenSTA 是一个专用于 B/S 结构的、免费的性能测试工具。它的优点除了免费、源代码开放外,还能对录制的测试脚本按照自己的语法规则进行编辑,具有与一般商业测试工具同样的灵活性。测试工程师在录制完测试脚本后,只需要了解 OpenSTA 脚本语言的语法规则,就可以对测试脚本进行编辑,以便于执行性能测试时能够获得所需要的测试数据,使脚本的业务仿真程度更高。OpenSTA 也提供了较为丰富的图形化测试结果,大大提高了测试报告的可读性。

OpenSTA 由以下三部分组成:

1) OpenSTA Commander

OpenSTA 的总控制台,负责测试脚本的录制,性能数据的监视,测试场景的部署、执行等。由以下三部分组成。

- Scripts:脚本的录制与调试;

- Collectors：监视性能数据的添加；
- Tests：测试场景部署、执行，测试结果。

2）OpenSTA Name Server

CORBA(Common Object Request Broker Architecture，公共对象请求代理体系结构)背景处理器，保证 OpenSTA 各个组成部分之间的交流，同时也是负载生成器的重要组件。OpenSTA Name Server 正常情况下随操作系统启动而自动运行，并以系统图表的形式显示在任务栏右下角。

3）Web Relay Daemon

Web 传送器，用于分布式结构的软件系统测试时的通信。

17.3.2 OpenSTA 的应用

1. 应用概述

1）OpenSTA 启动

选择"开始"→"程序"→OpenSTA→OpenSTA Commander 命令，启动 OpenSTA（如图 17-19 所示）。

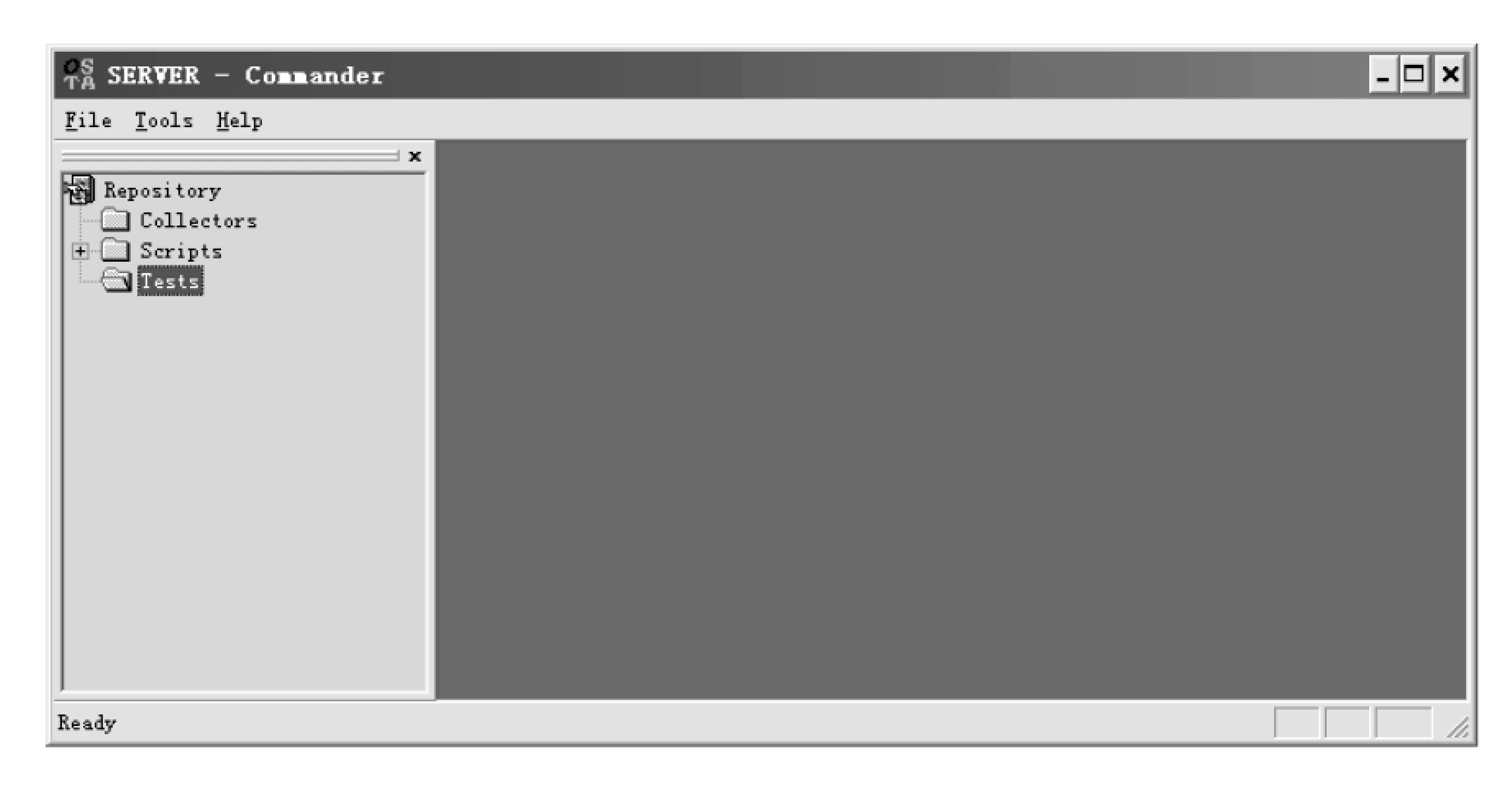


图 17-19 OpenSTA 主界面

OpenSTA Commander 启动后会默认打开一个 Repository（工作空间），Repository 下有三个默认的文件夹，分别是 Collectors、Scripts 和 Tests。其中，Collectors 是收集器，在这里我们可以添加要监控和收集数据的操作系统计数器；Scripts 是测试脚本开发工具，支持录制开发模式；Tests 用于部署测试场景（载入测试脚本、设定虚拟用户等），运行测试脚本，记录测试结果。

2）OpenSTA 主要工作窗口介绍

（1）Script Modeler。在图 17-19 中 Script 目录下新建一个 Script（假设命名为 myScript），鼠标双击 myScript 即可打开 Script Modeler 窗口（如图 17-20 所示）。Script Modeler 主要包含以下功能区：

- Menu Bar(菜单栏)：包含脚本开发的各项控制和设置；
- File Toolbar(文件工具栏)：新建、保存和关闭脚本等功能控制；
- Record Toolbar(录制工具栏)：脚本录制的相关控制快捷按钮；
- Variable Toolbar(变量工具栏)：为脚本设置变量的有关控制快捷按钮；
- Script Editor(脚本编辑器)：用以开发测试脚本；
- Analysis(分析)：对某项操作的代码结构分析和服务端、客户端的头数据分析；
- Script Debug(脚本调试信息输出栏)：用以输出脚本调试、回放信息等。

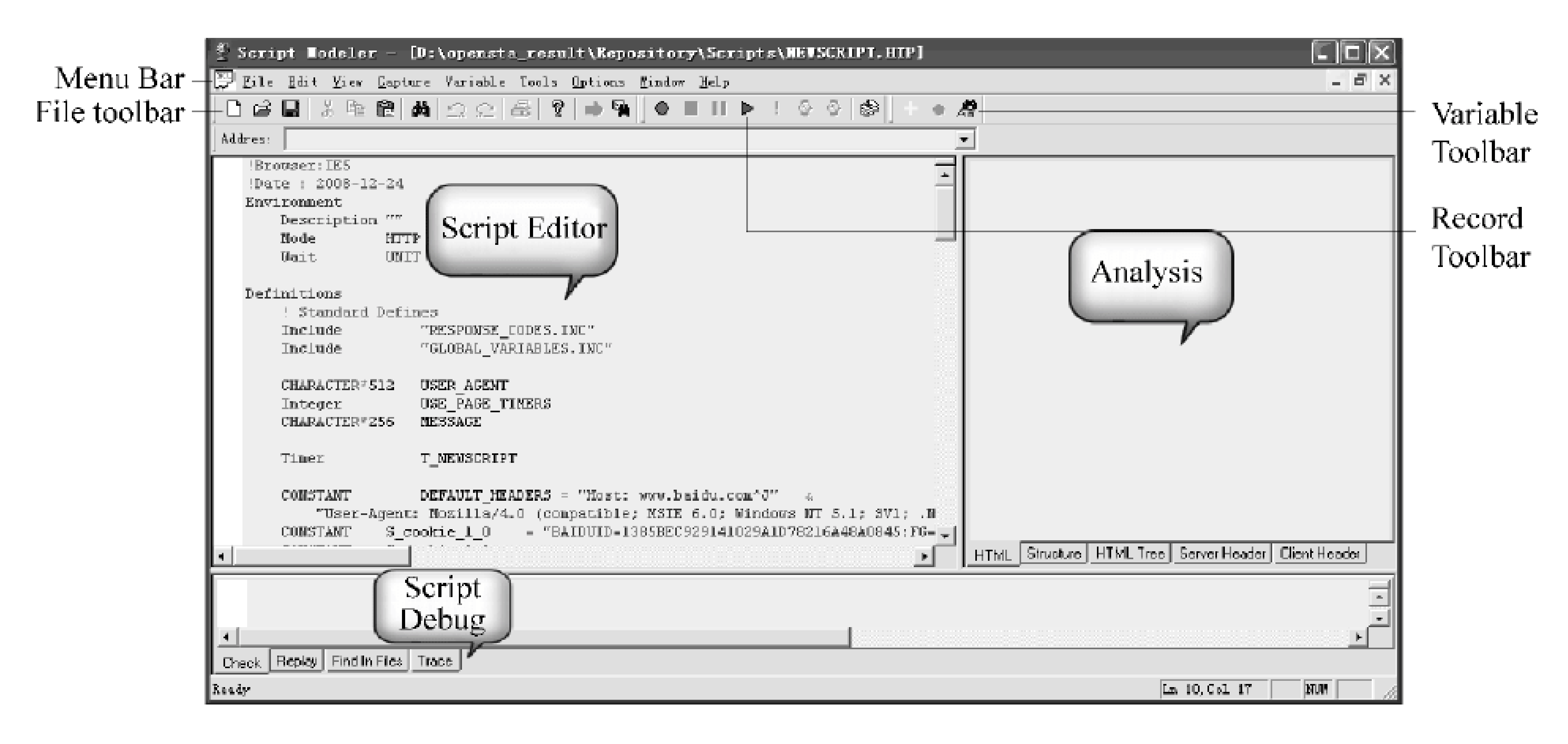


图 17-20　Script Modeler 主界面

(2) OpenSTA Commander-Collectors。在 Collectors 下新建一个收集器(默认命名为 NEWCOLLECTOR)，鼠标双击 NEWCOLLECTOR 即可打开 Collectors 窗口(如图 17-21 所示)。Collectors 窗口比较简单，右边区域用来添加、删除有关性能计数器。Collectors 支持 SNMP 和 NT Performance，其中 SNMP 是简单网络管理协议，NT Performance 可以监控 Windows 系列的服务器性能，如处理器、内存、磁盘等众多指标参数。

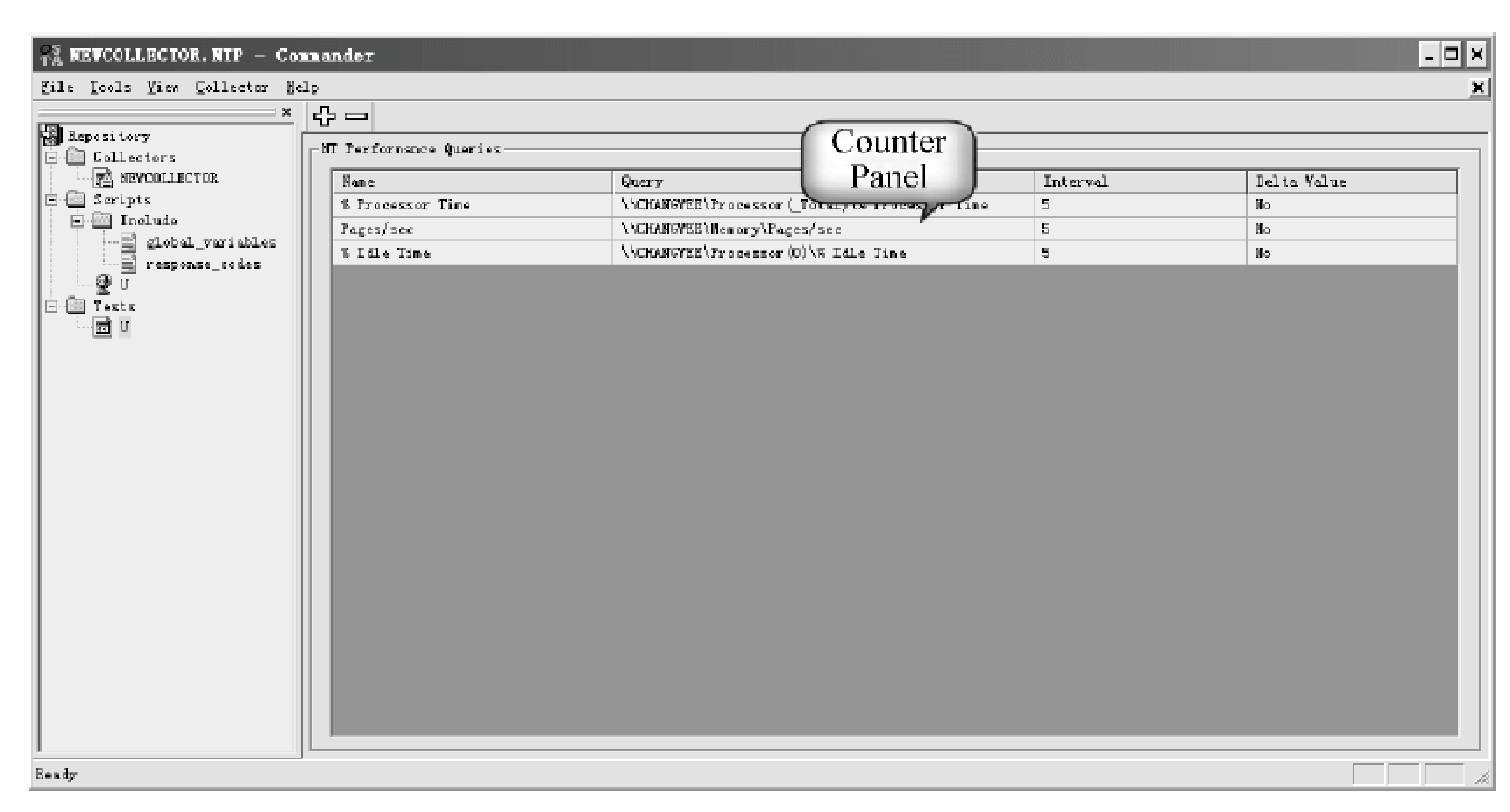


图 17-21　Collectors 主界面

(3) OpenSTA Commander-Tests。在 Tests 目录下新建一个 Tests(默认命名为 NEWTEST),鼠标双击 NEWTEST 即可打开 Tests 窗口(如图 17-22 所示)。Tests 窗口主要包含以下功能区域:

- Menu Bar(菜单栏):创建、关闭 Tests;Tests 执行控制等;
- Test Excute Toolbar(测试执行控制工具栏):测试执行控制快捷按钮;
- Configuration Tab(配置):配置面板包含 Scenario Schedule 和 Load Mode,主要用来部署测试场景、设定加压方式;
- Monitoring(监视器):监视场景的运行情况;
- Results(测试结果):采集分析并显示测试结果。

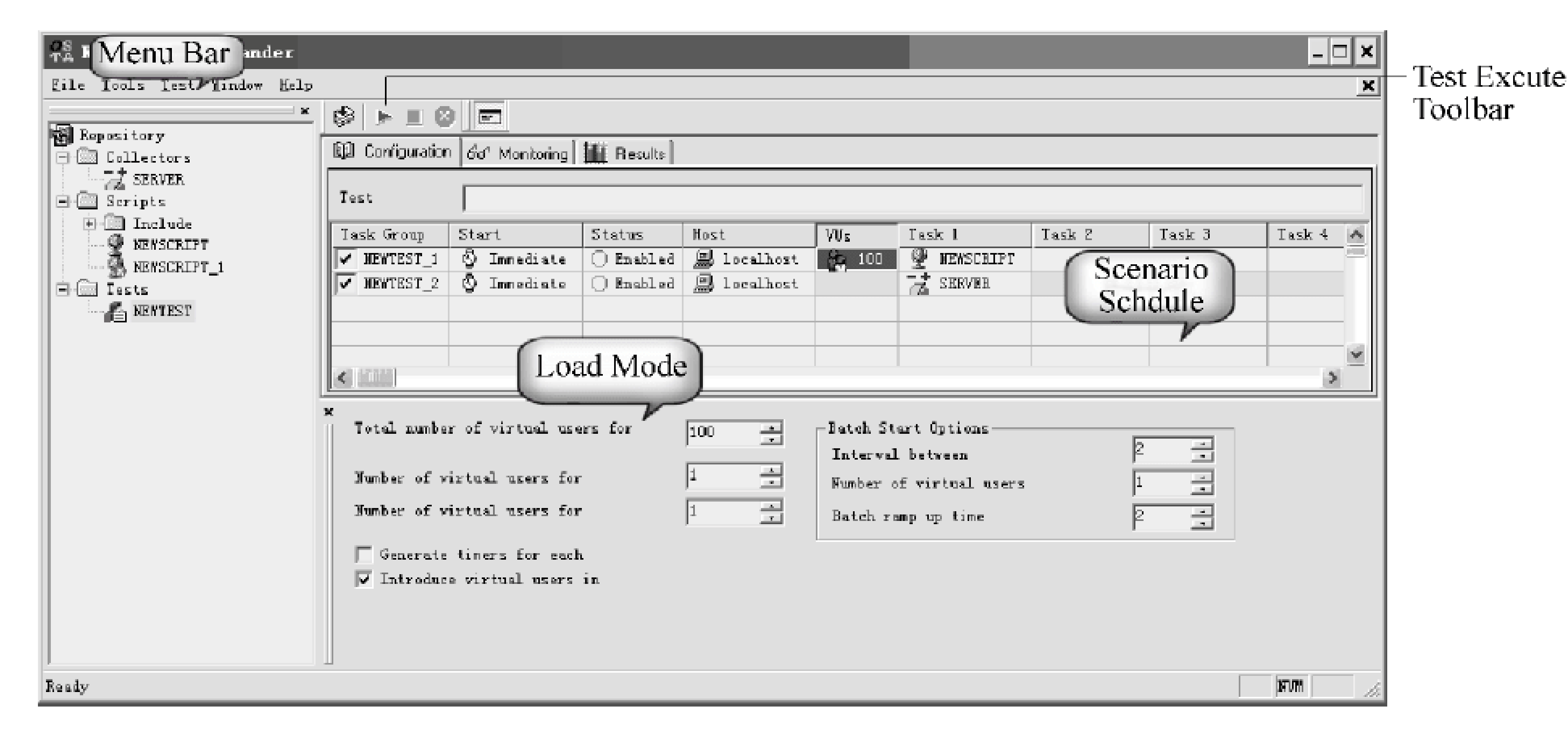


图 17-22 Tests 主界面

2. OpenSTA 测试流程

应用 OpenSTA 进行性能测试,一般遵循以下流程:

1) 准备测试脚本

根据指定的测试用例,准备相应的测试脚本,同时对测试脚本进行增强,如参数化、关联等。

2) 添加监视计数器

添加有关的性能计数器,以便进行测试结果分析。

3) 创建测试场景

将准备的测试脚本、性能计数器文件拖到 Tests 中,并分配各测试脚本的虚拟用户数、执行过程中的相关参数设定,如迭代执行时间间隔、用户加载方式等。

4) 运行测试场景

单击 OpenSTA 的场景运行按钮,运行创建的各种测试场景。可以在各场景运行过程中,监视各种测试数据。

5) 分析测试结果

根据测试得到的数据结果,进行深入的分析,得到测试结果。

3. OpenSTA 应用举例

首先检查 OpenSTA Name Server 是否正常启动，否则首先应启动 OpenSTA Name Server，然后打开 OpenSTA Commander(如图 17-19 所示)。在 OpenSTA Commander 中右击 Scripts，新建脚本，输入脚本名称，然后双击新脚本，打开 Script Modeler 窗口。

在这个窗口中可以看到 OpenSTA 有内嵌的浏览器，在 Options 菜单中设置 Browser 和 Gateway。一般情况下，保持 Gateway 的缺省设置。采用录制的方式取得测试脚本，单击工具栏中的红色按钮，即可开始脚本录制，OpenSTA 自动打开指定的浏览器，在浏览器中输入要测试的网站，并进行相应的操作，OpenSTA 会自动记录我们的操作过程。录制完毕后，按工具条上的停止按钮，或直接退出浏览器，即可停止脚本的录制。停止录制后，OpenSTA 自动生成测试脚本，并显示在 Script Modeler 窗口中(如图 17-23 所示)。

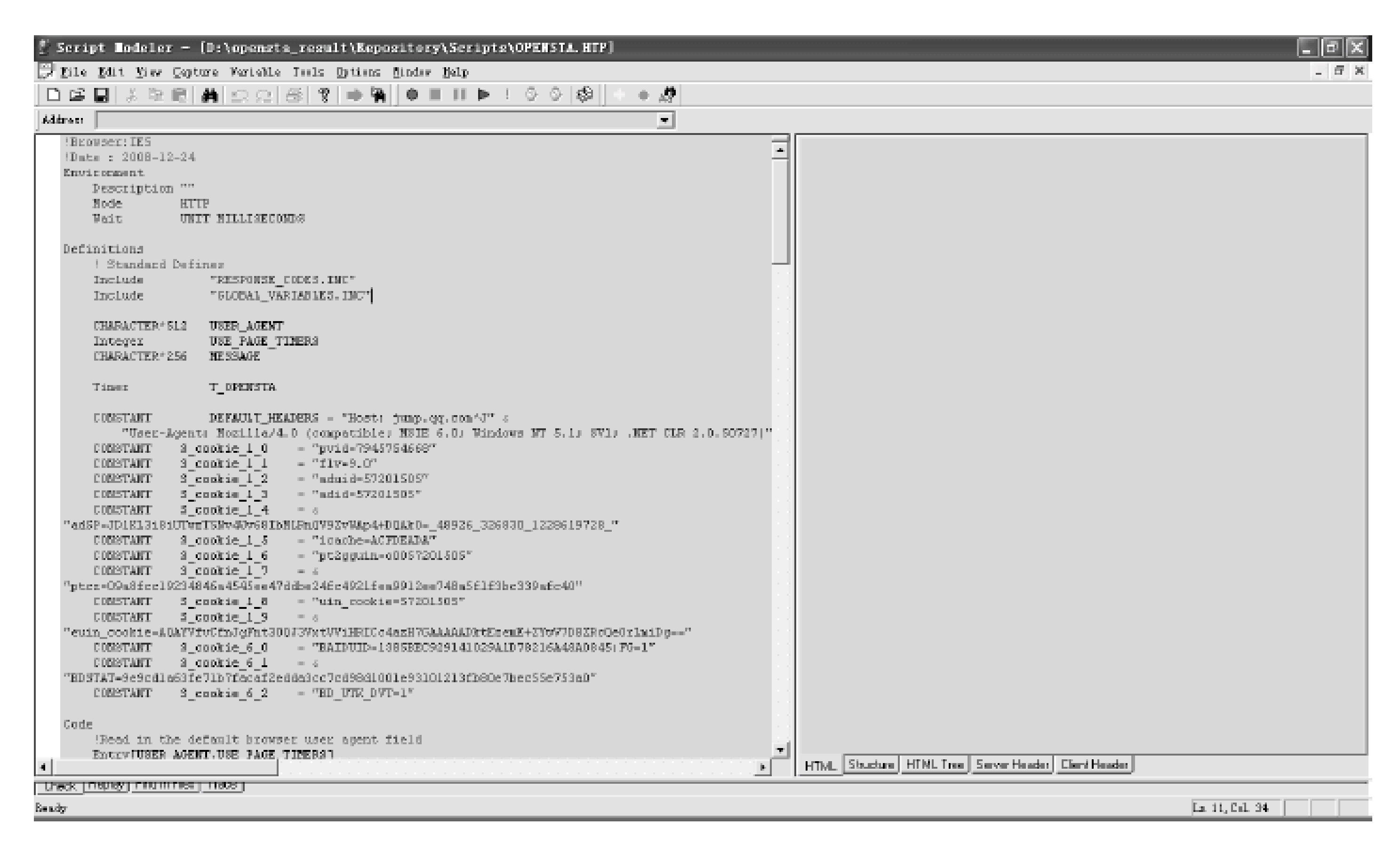


图 17-23　Script Modeler

保存测试脚本后，回到 Commander 界面，右击 Collectors，新建一个计数器收集器，这里我们选择 NT Performance 来监视 Windows 的资源，并输入收集器的名字，然后双击该收集器，添加相应的计数器(如图 17-24 所示)。



图 17-24　添加计数器

保存新建的收集器后，开始新建一个测试。在 Commander 界面中，右击 Tests，新建一个测试，输入测试的名字，然后双击该名字，进入测试场景部署界面，将刚才建立的测试脚本和性能计数器收集器拖入该页面中的 Task 下，并进行相应的参数设置（如图 17-25 所示）。

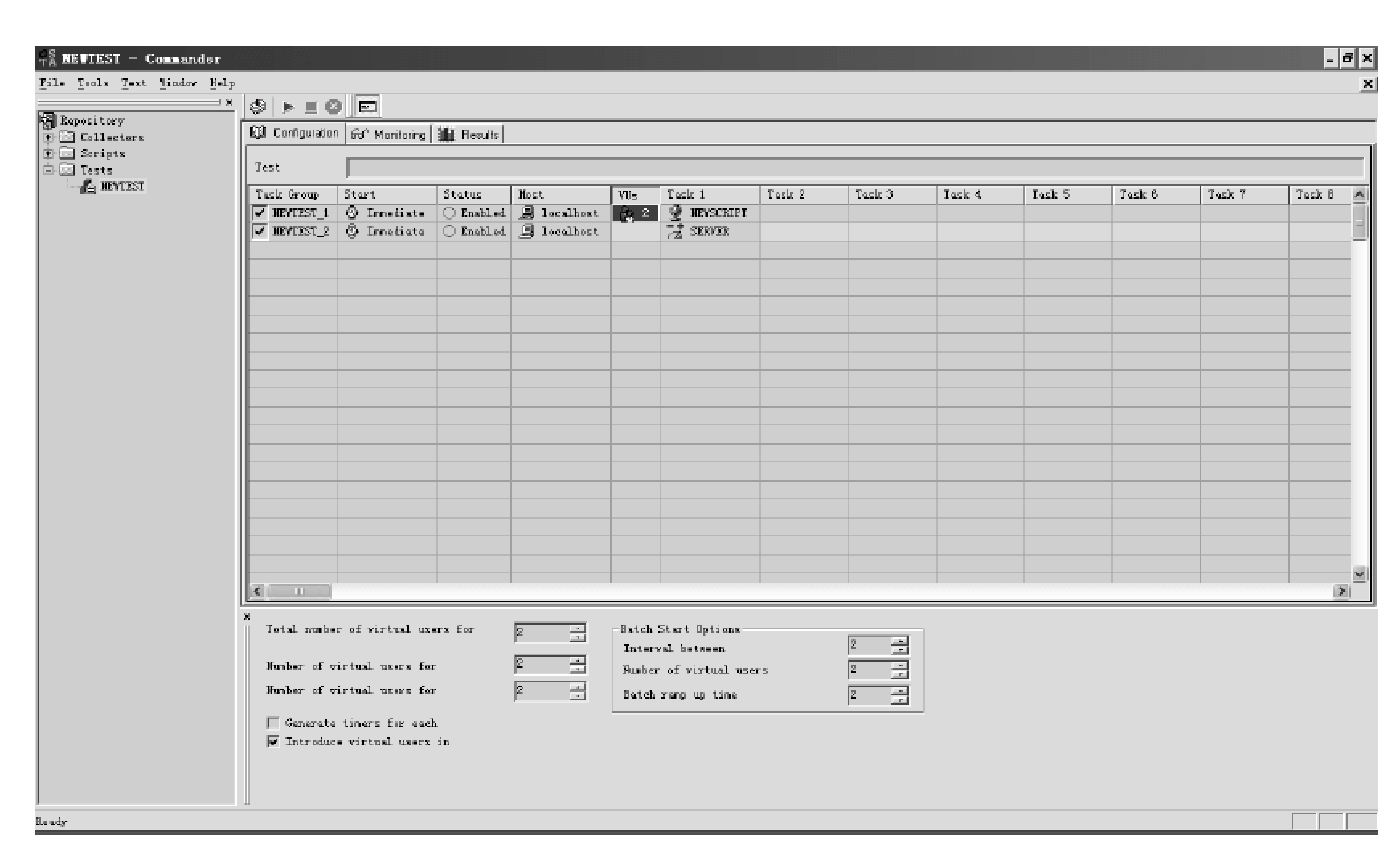


图 17-25 OpenSTA 测试场景

场景参数设置完毕，单击运行按钮，执行测试。测试中，可以切换到 Monitoring，在这里可以监视部分测试执行情况的数据（如图 17-26 所示）。测试运行完毕，可以切换到 Results，查看测试结果（如图 17-27 所示），通过对测试结果的综合分析，得出测试结论。

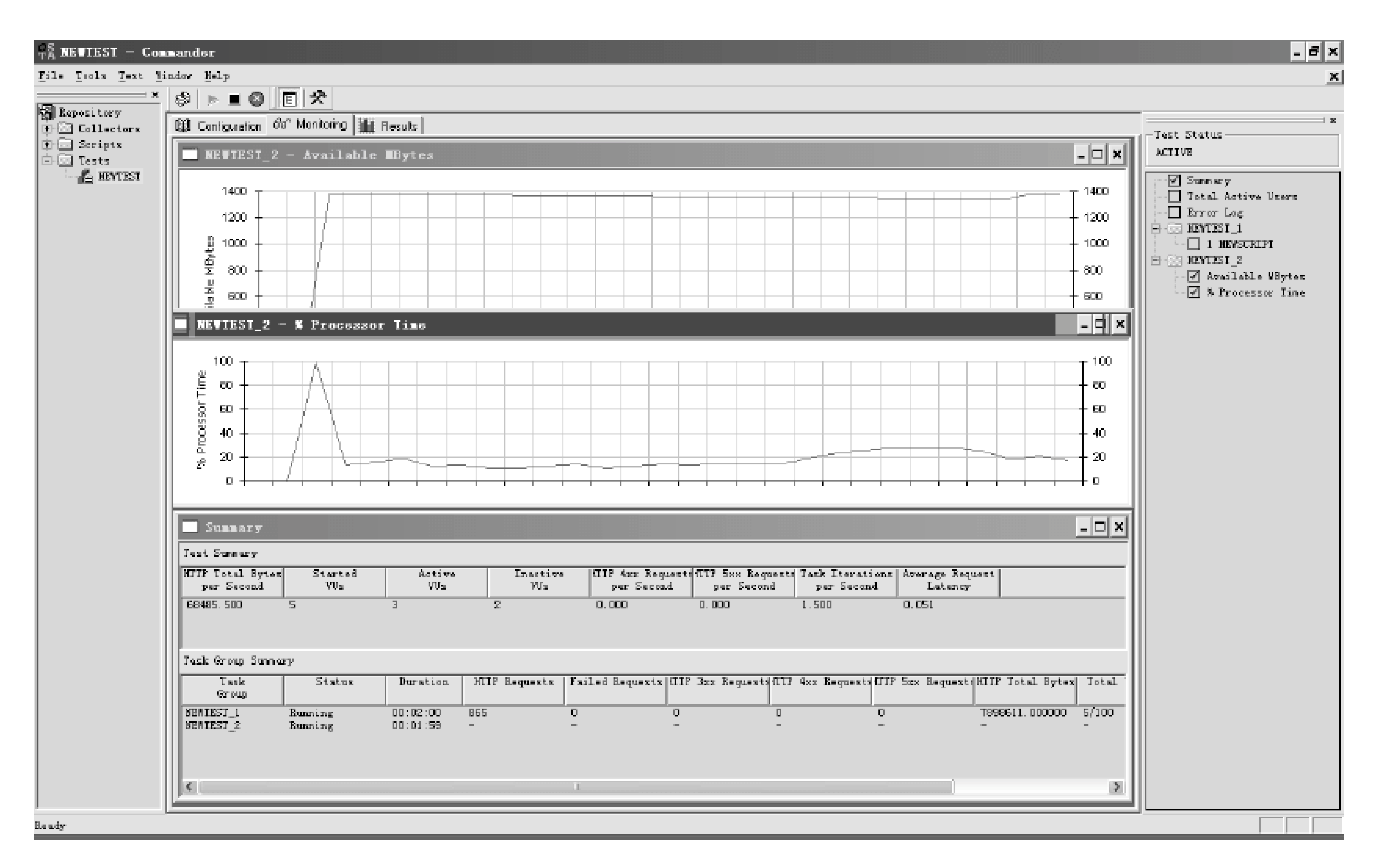


图 17-26 OpenSTA Monitoring

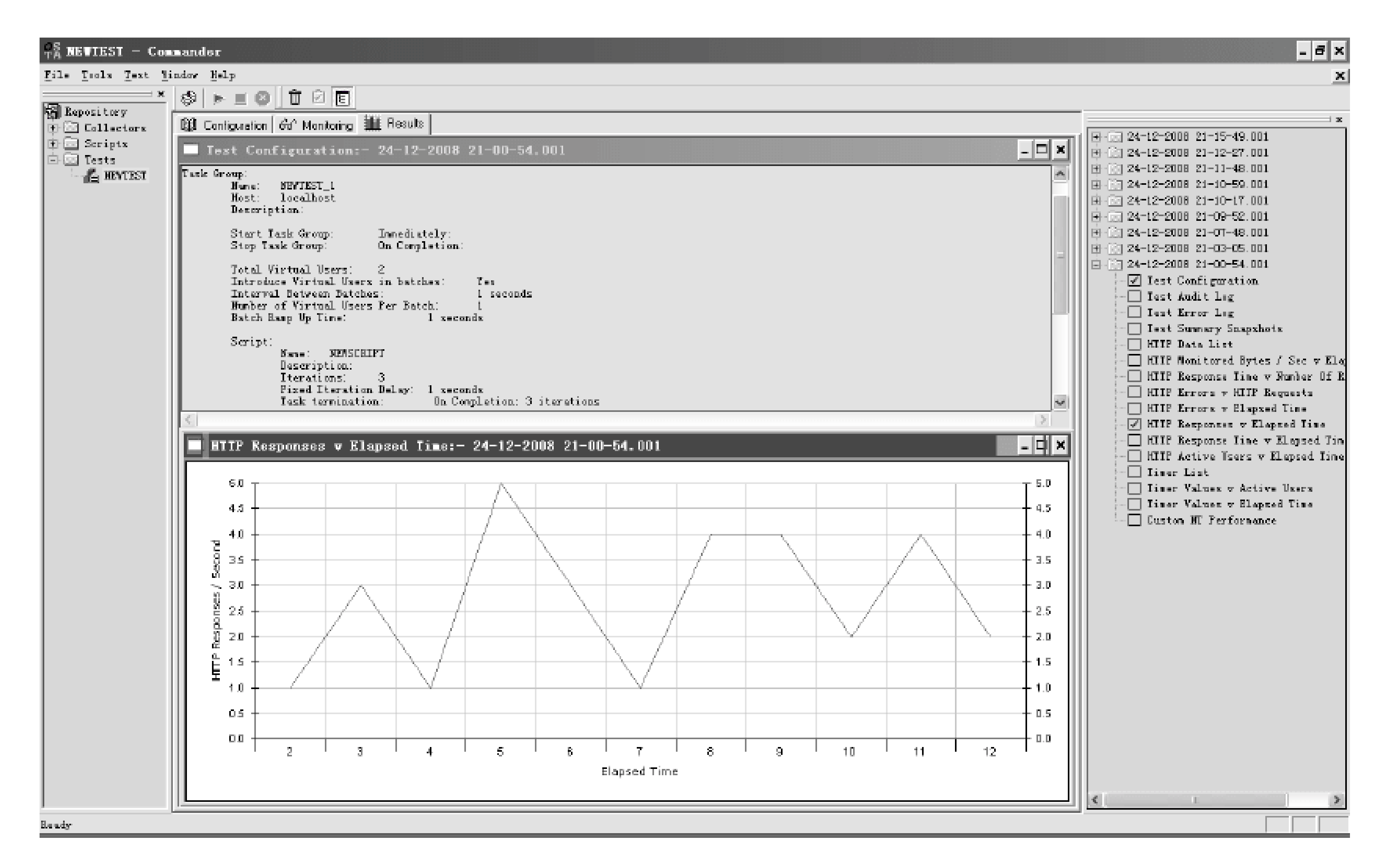


图 17-27 OpenSTA Results

本章小结

本章简单介绍了软件性能测试，以及一款典型的商业性能测试工具 LoadRunner 和一款出色的免费开源测试工具 OpenSTA 的使用方法、常见问题及解决办法，并以实例的形式示范了这两款测试工具的使用过程。

性能测试一般包括如下几个步骤：分析测试需求、确定测试策略、制定测试方案（计划）、开发测试脚本、部署、执行测试场景、分析测试结果、编写测试报告。

LoadRunner 是一款功能强大的商业性能测试工具，其价格非常的昂贵，是目前应用最为广泛的性能测试工具之一。LoadRunner 由四部分组成：脚本生成器（VU Generator）、控制台（Controler）、结果分析器（Analysis）、负载生成器（Agent）。

OpenSTA 是一个专用于 B/S 结构的性能测试工具，免费开源，有兴趣的读者可以进一步研究。

CHAPTER

第 18 章 测试管理工具

测试管理工具用于对测试进行管理。一般而言,测试管理工具实现对测试计划、测试用例、测试实施以及软件缺陷的跟踪管理。测试管理工具一般具有 C/S 或者 B/S 架构,能够让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库,在任何可以联通网络的地方都能进行信息交互。

当前市面上,测试管理工具非常丰富,其中比较有代表性的有 HP-Mercury 公司的 TestDirector、Rational 公司的 Test Manager、Compureware 公司的 TrackRecord 等软件。本章主要介绍一下 HP 公司的 TestDirector 和一款小巧的开源软件 Bugzilla。

18.1 TestDirector

18.1.1 概述

TestDirector 是 HP 公司一个测试管理工具,是业界第一个基于 Web 的测试管理系统,它可以在您公司内部或外部进行全球范围内测试管理。通过在一个整体的应用系统中集成了测试管理的各个部分,包括需求管理,测试计划,测试执行以及错误跟踪等功能,TestDirector 极大地加速了测试过程。

TestDirector 能消除组织机构间、地域间的障碍。它能让测试人员、开发人员或其他相关人员通过一个中央数据仓库,在不同地方就能交互测试信息。TestDirector 将测试过程流水化——从测试需求管理,到测试计划,测试日程安排,测试执行到出错后的错误跟踪——仅在一个基于浏览器的应用中便可完成,而不需要每个客户端都安装一套客户端程序。

TestDirector 具有如下功能特点。

1. 需求管理

程序的需求驱动整个测试过程。TestDirector 的 Web 界面简化了这些需求管理过程,以此您可以验证应用软件的每一个特性或功能是否正常。通过提供一个比较直观的

机制将需求和测试用例、测试结果和报告的错误联系起来，从而确保能达到最高的测试覆盖率。

2. 计划测试

测试计划的制订是测试过程中至关重要的环节。它为整个测试提供了一个结构框架。TestDirector 的 Test Plan Manager 在测试计划期间，为测试小组提供一个关键要点和 Web 界面来协调团队间的沟通。

Test Plan Manager 指导测试人员如何将应用需求转化为具体的测试计划。这种直观的结构能帮助测试人员定义如何测试应用软件，从而能组织起明确的任务和责任。Test Plan Manager 提供了多种方式来建立完整的测试计划。测试人员可以从草图上建立一份计划，或根据 Requirements Manager 所定义下的应用需求，通过 Test Plan Wizard 快捷地生成一份测试计划。如果测试人员已经将计划信息以文字处理文件形式，如 Microsoft Word 方式储存，可以再利用这些信息，并将它导入到 Test Plan Manager。它把各种类型的测试汇总在一个可折叠式目录树内，可以在一个目录下查询到所有的测试计划。例如，你可以将人工和自动测试，如功能性的，还原和负载测试方案结合在同一位置。

Test Plan Manager 还能进一步的帮助测试人员完善测试设计和以文件形式描述每一个测试步骤，包括对每一项测试，用户反应的顺序，检查点和预期的结果。TestDirector 还能为每一项测试连加附属文件，如 Word、Excel、HTML ，用于更详尽地记录每次的测试计划。

TestDirector 还能简化将人工测试切换到自动测试脚本的转化，并可立即启动测试设计过程。

3. 安排和执行测试

一旦测试计划建立后，TestDirector 的测试实验室管理为测试日程制订提供一个基于 Web 的框架。它的 Smart Scheduler 根据测试计划中创立的指标对运行着的测试执行监控。当网络上任何一台主机空闲，测试可以彻夜执行于其上。Smart Scheduler 能自动分辨是系统还是应用错误，然后将测试重新安排到网络上的其他机器。

4. 缺陷管理

TestDirector 的出错管理直接贯穿作用于测试的全过程，以提供管理系统终端-终端的出错跟踪——从最初的问题发现到修改错误再到检验修改结果。由于同一项目组中的成员经常分布于不同的地方，TestDirector 基于浏览器的特征，使出错管理能让多个用户随时随地都可通过 Web 查询出错跟踪情况。利用出错管理，测试人员只需进入一个 URL ，就可汇报和更新错误，过滤整理错误列表并作趋势分析。在进入一个出错案例前，测试人员还可自动执行一次错误数据库的搜寻，确定是否已有类似的案例记录。这一查询功能可避免重复劳动。

5. 图形化和报表输出

TestDirector 常规化的图表和报告可在测试的任一环节帮助用户对数据信息进行分析。TestDirector 还以标准的 HTML 或 Word 形式提供生成和发送正式测试报告的一

种简单方式。测试分析数据还可简便地输入到一种工业标准化的报告工具，如 Excel、ReportSmith、CrystalReports 和其他类型的第三方工具。

18.1.2　TestDirector 的应用

1. 应用概述

1）启动 TestDirector

打开 Web 浏览器，输入 TestDirector 所在的 URL（http://[Serverame]/tdbin/default.htm），TestDirector 的首页将被打开（如图 18-1 所示）。



图 18-1　TestDirector 首页

单击左侧的 TestDirector，进入 TestDirector 的主界面（如图 18-2）。选择域和工程，输入用户名和密码，登录进入 TestDirector 主界面。这里我们选用默认的域和工程，采用 TestDitector 默认的超级用户 Admin，输入密码，单击 Login，登录 TestDirector。



图 18-2　TestDirector 主界面

单击 TestDirector 首页左侧的 Site Administrator，输入密码（默认为空），进入 TestDirector 的管理界面（如图 18-3 所示）。

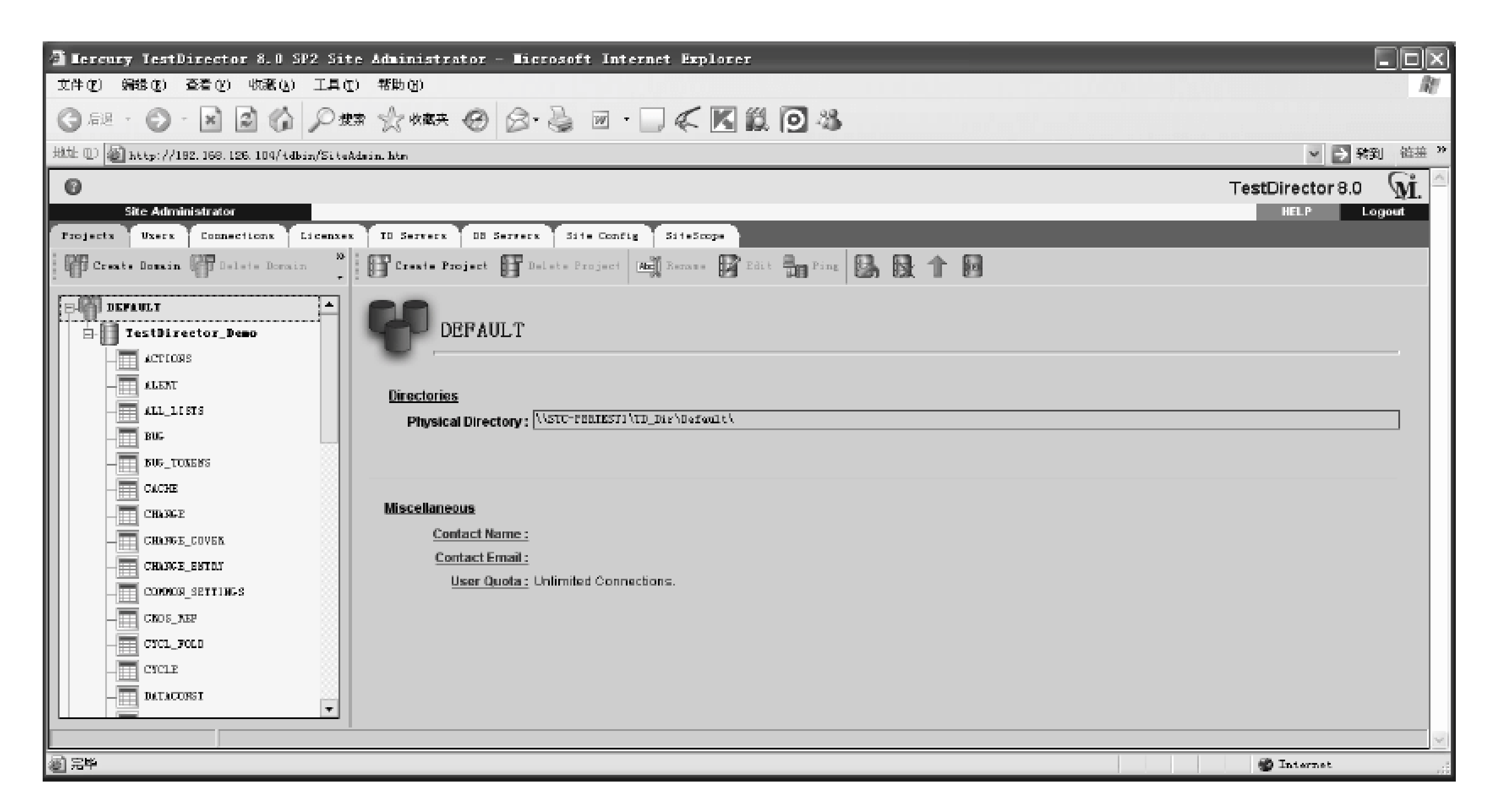


图 18-3　TestDirector 管理界面

2）TestDirector 主窗口

TestDirector 的主窗口由以下几部分构成，包括（如图 18-4 所示）：

- TestDirector Toolbar：显示工程常用的命令。
- Menu Bar：显示各模块常用的命令，菜单名称随你选择的模块名称不同而改变。
- Module Toolbar：显示模块常用的命令。
- Tools Button：显示常用的工具。

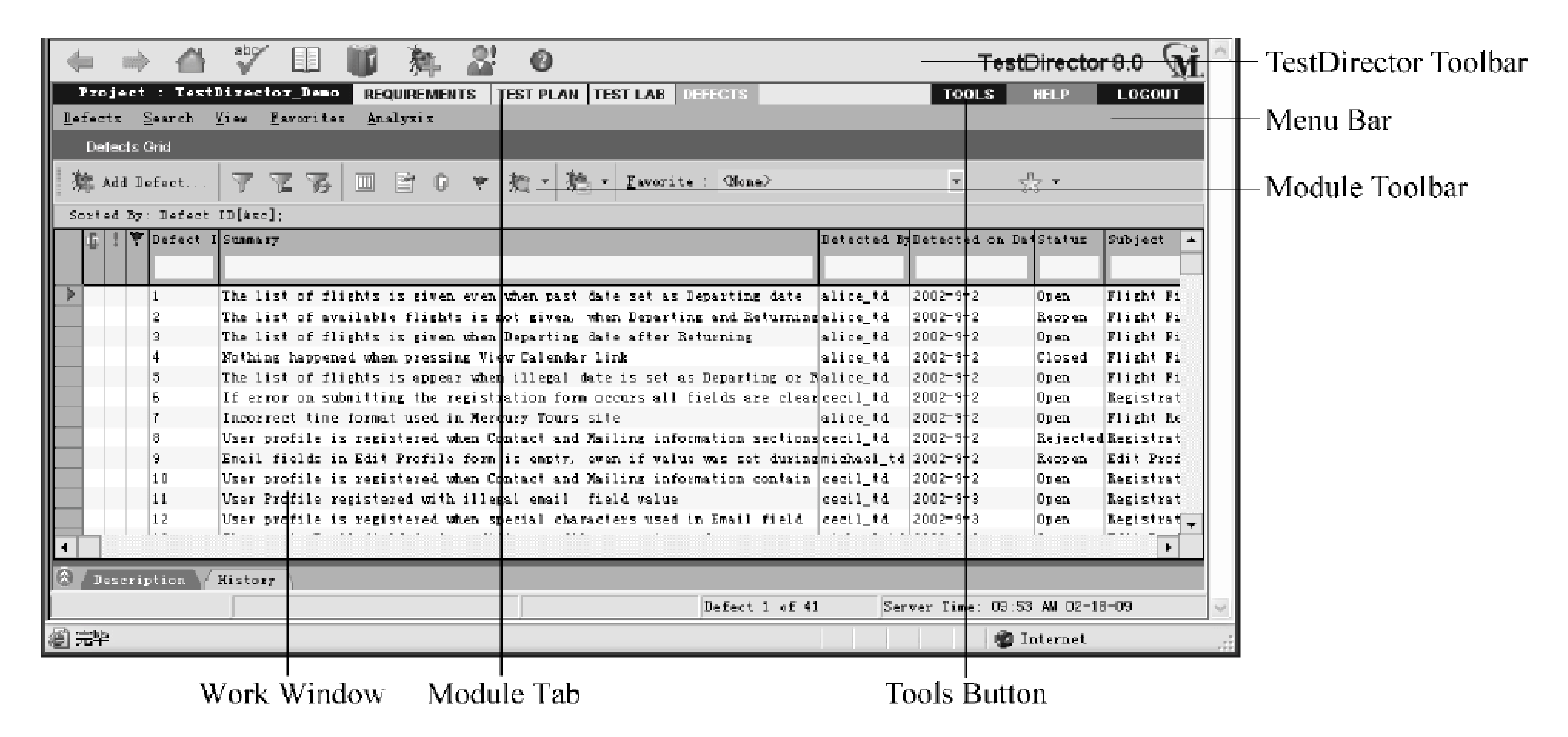


图 18-4　TestDirector 工作界面构成

2．TestDirector 实践和应用举例

在本书所提供的案例“人力资源管理系统（HRMIS）”中，整个测试过程也是借由 TestDirector 来开展的，下面我们来具体看一下这个应用过程。

1）项目管理库环境配置

项目管理环境指由项目负责人提出的涉及测试管理的一系列管理要素，一般涉及以下几个方面（如图 18-5 所示）：

（1）项目标识。TestDirector 有域（Domain）和项目（Project）两个概念，即支持产品域和项目的两级定义。通过这两个类别基本上可以满足大中型企业的产品管理需求。对于域这个概念，通常可以有以下几种理解：

① 视产品域为部门，即每个产品研发单位在 TestDirector 中开设一个域，比如 BOSS 研发部，我们定义其域名称为 PD_BOSS，当然 TestDirector 是可以定义中文名称的域名称的。

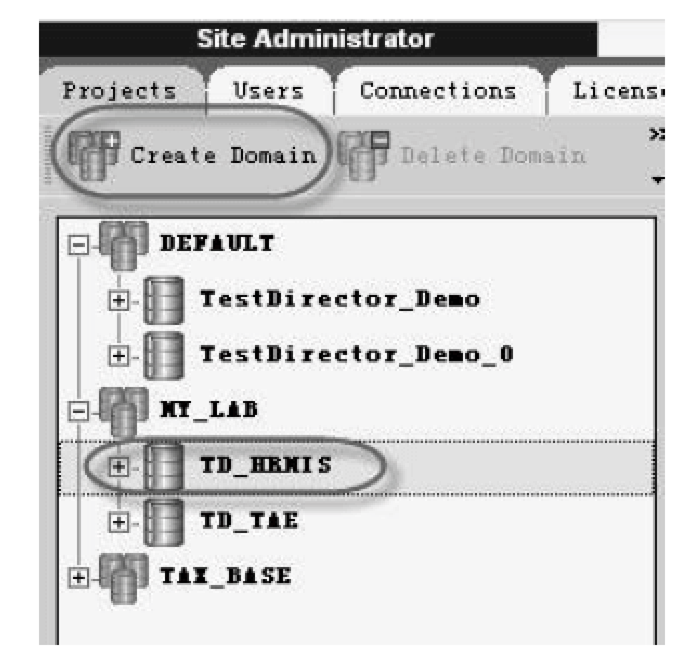


图 18-5 TestDirector 域/项目规划

② 视产品域为一个产品线，这个适用于以产品为中心的开发组织，这样可以将一条产品线的多个分支统一归到所属产品线中进行集中管理。

③ 视产品域为一个市场区域，这种方式适用于以市场为中心的开发组织，这样的好处是可以将同一地区的不同类型的项目归到一个区域进行同一管理。

TestDirector 的项目（Project）直接对应实体项目，比如本例中“人力资源管理系统”，我们可以定义项目为 T_HRMIS。项目标识最好按照统一的标准命名，这样方便查找和识别。

（2）项目成员及其权限分配。TestDirector 有完善的权限控制模块，可以实现岗位定义和人员功能授权，基本上可以满足软件测试项目管理的要求。

在本例中，我们有三名测试人员参与 T_HRMIS 的测试，在项目库创建之初，将对应这三个测试人员的系统用户添加到 T_HRMIS 中（如图 18-6 所示），并分配岗位为 QATester。假如系统原有岗位不能满足权限分配要求，则可以通过项目管理模块新增岗位，在本例中我们新增了一个角色 Tester（如图 18-7 所示），选中 Tester，单击 Change 可

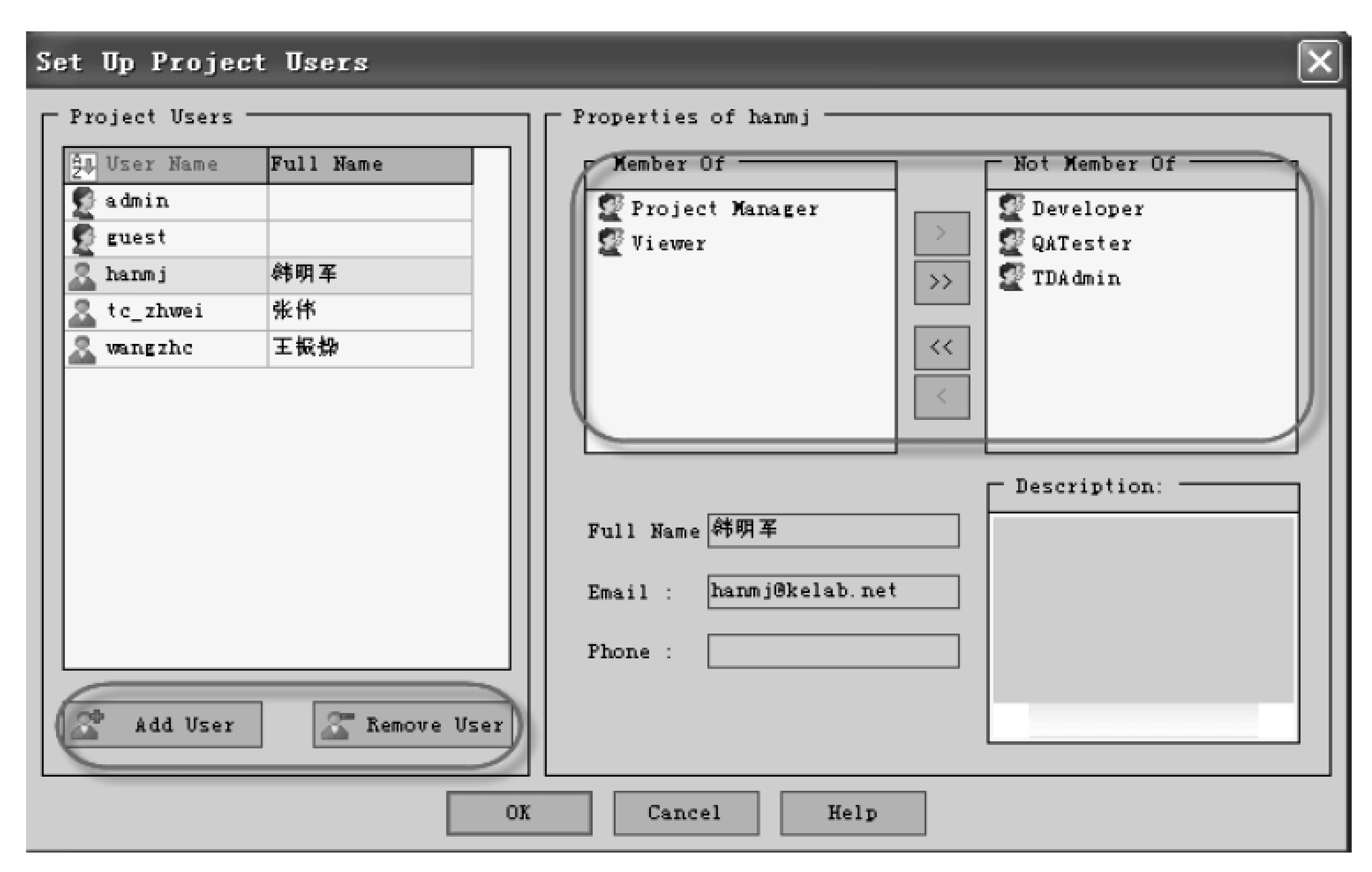


图 18-6 项目成员设置

以设置该岗位的系统使用权限(如图 18-7 所示)。

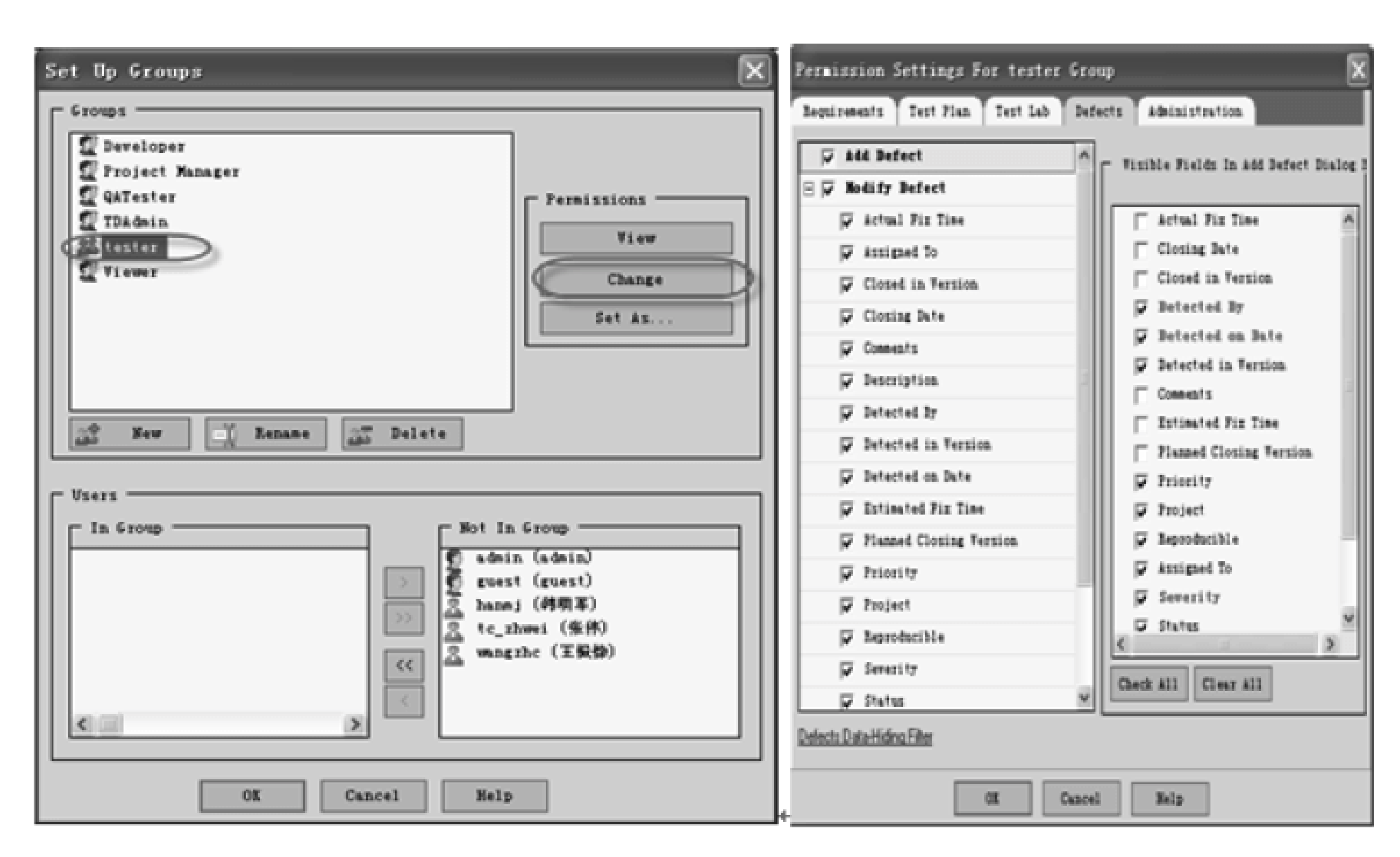


图 18-7　权限设置

(3) 缺陷属性定义。在使用 TestDirector 管理测试项目前,首先应该对测试管理配置项进行规划,比如缺陷库字段定义(涉及字段数目、字段标识、数据类型等)。TestDirector 支持字段自定义,可以通过灵活的字段定义满足项目的管理要求。在本例中我们在测试需求库中增加了 5 个自定义字段(如图 18-8 所示),分别是计划开始、计划完成时间、任务状态、实际开始和实际完成时间。TestDirector 支持设置字段的数据类型和应用方式(如手工输入、下拉框选择等界面表现形式),相当灵活。

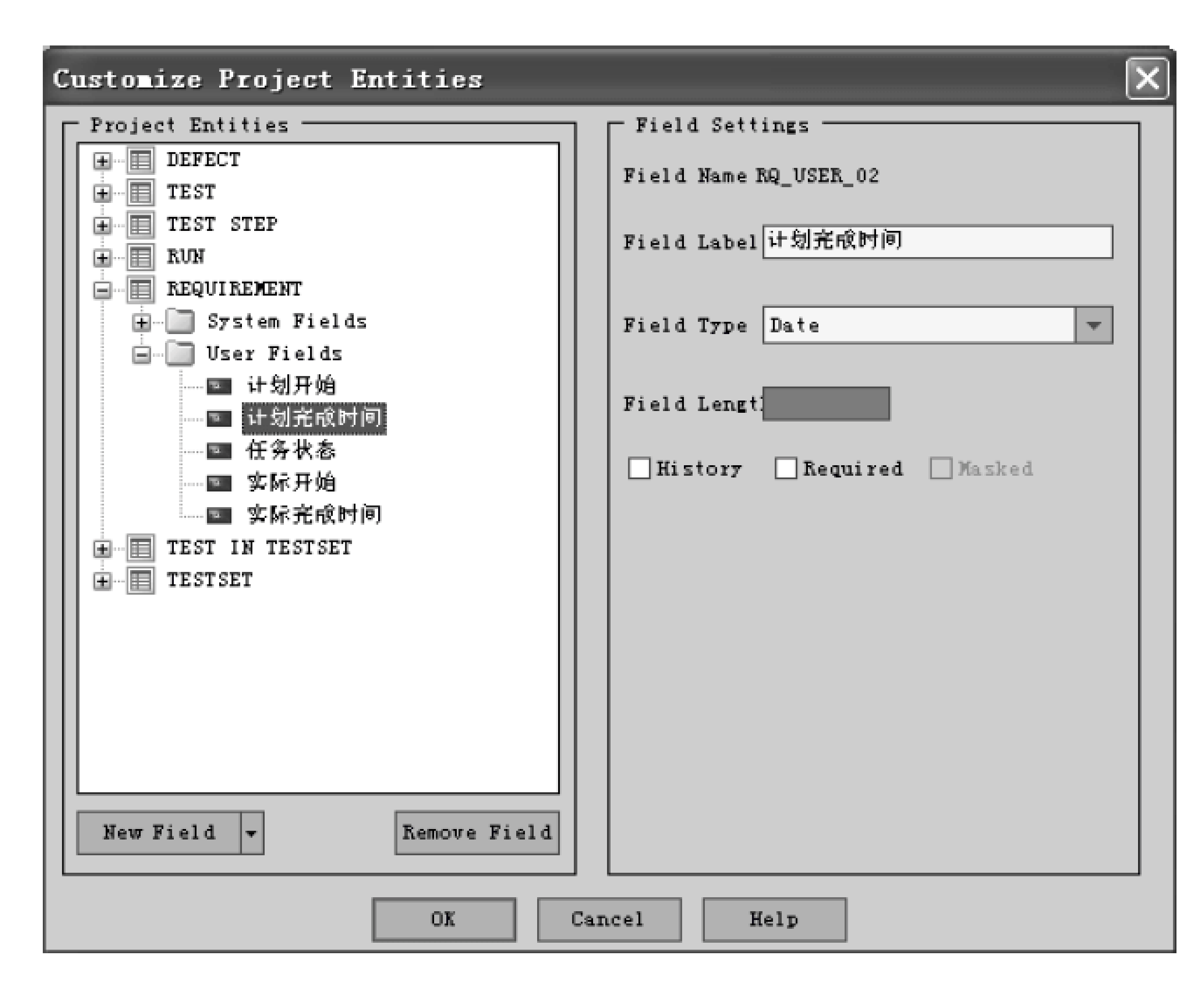


图 18-8　自定义字段

2) 使用 TestDirector 管理测试需求

使用 TestDirector 可以方便地管理测试需求,测试需求以分条目的形式添加到

Reuirements 栏的需求列表中，需求之间还可以设置分级，使其关系更加清楚明了（如图 18-9 所示）。

Name	Direct Cover Status	ReqID	Author	Reviewed	Creation Time	Creation Date
业务功能	Not Covered	[RQ0001]	tc_zhwei	Not Reviewed	16:48:45	2008-12-15
New Requirement	Not Covered	[RQ0067]	tc_zhwei	Not Reviewed	15:47:47	2009-1-6
Code Improve	Not Covered	[RQ0055]	tc_zhwei	Not Reviewed	16:00:42	2008-12-24
测试需求	Not Covered	[RQ0009]	tc_zhwei	Not Reviewed	10:29:55	2008-12-16
列表采用第三方控件，对于大数据量...	Not Covered	[RQ0010]	tc_zhwei	Not Reviewed	10:30:58	2008-12-16
员工信息管理	Not Covered	[RQ0068]	tc_zhwei	Not Reviewed	15:48:43	2009-1-6
基本信息管理	Not Covered	[RQ0069]	tc_zhwei	Not Reviewed	15:52:16	2009-1-6
工作合同管理	Not Covered	[RQ0070]	tc_zhwei	Not Reviewed	15:52:35	2009-1-6
工作履历	Not Covered	[RQ0072]	tc_zhwei	Not Reviewed	15:53:17	2009-1-6
培训管理	Not Covered	[RQ0071]	tc_zhwei	Not Reviewed	15:52:50	2009-1-6

图 18-9　TestDirector 测试需求

TestDirector 中关于测试需求的管理类似于测试矩阵的模式。可以与测试用例建立关联关系，通过用例的设计执行情况，间接考查测试需求的覆盖。

3）使用 TestDirector 管理测试用例

使用 TestDirector 可以在 Test Plan 栏进行测试用例的设计和编写（如图 18-10 所示）。TestDirector 可以与 HP 公司的其他测试工具，如我们前面介绍的 WinRunner、QTP、LoadRunner 等很好地集成，可以直接使用 TestDirector 调用这几款测试工具的自动化测试脚本，也就是说，TestDirector 的测试用例设计支持手工编写和自动化测试脚本多种方式，表 18-1 列出了 TestDirector 的所有支持的测试用例设计编写方式。



图 18-10　测试用例的编写

表 18-1　TestDirector 的用例编写方式

测试类型	描　　述
MANUAL	手动测试
WR-AUTOMATED	通过 WinRunner 执行
VAPI-TEST	通过 Visual API 执行。TestDirector 的 API 执行工具，可以创建和运行 C Scripts
LR-SCENARIO	LoadRunner 的场景，通过 LoadRunner 执行
QUICKTEST-TEST	通过 QuickTest Professional 执行或通过 Astra QuickTest 执行

续表

测试类型	描　　述
ALT-TEST	通过 Astra LoadTest 执行。Astra LoadTest 是早期 Mercury Interactive 公司为 Web 应用程序开发的负载测试工具
ALT-SECNARIO	通过 Astra LoadTest 执行的场景
QTSAP-TESTCASE	通过 QuickTest Professional for MySAP. com Windows Client 执行。是早期 Mercury Interactive 公司为 MySAP. com 应用程序研制的功能测试工具，适用于 Windows 95、Windows 98、Windows 2000 和 Windows NT
XRUNNER	通过 XRunner 执行，XRunner 是早期 Mercury Interactive 公司为 X Windows 应用程序的自动化测试工具
VAPI-XP-TEST	用 Visual API-XP 创建
SYSTEM-TEST	它要求 TestDirector 去提供系统信息、捕获桌面图像或重启计算机

4）使用 TestDirector 管理软件缺陷

TestDirector 在软件缺陷管理模块做得非常出色，测试人员可以在测试过程的任何一个阶段录入所发现的缺陷，并完成整个缺陷生命周期的跟踪处理（如图 18-11 所示）。缺陷状态的变更受到严格控制，在 TestDirector 中，不同角色的人员对缺陷的状态的变更权限是不一样的，比如普通开发人员是不运行关闭或者删除缺陷的。

TestDirector 还提供了缺陷的属性自定义功能，有助于测试组织定义适合自己独特要求的缺陷属性。

作为业界知名的测试管理工具，TestDirector 提供了非常丰富的有关缺陷统计分析的内容，比如缺陷报告、缺陷趋势分析、缺陷状态统计等。

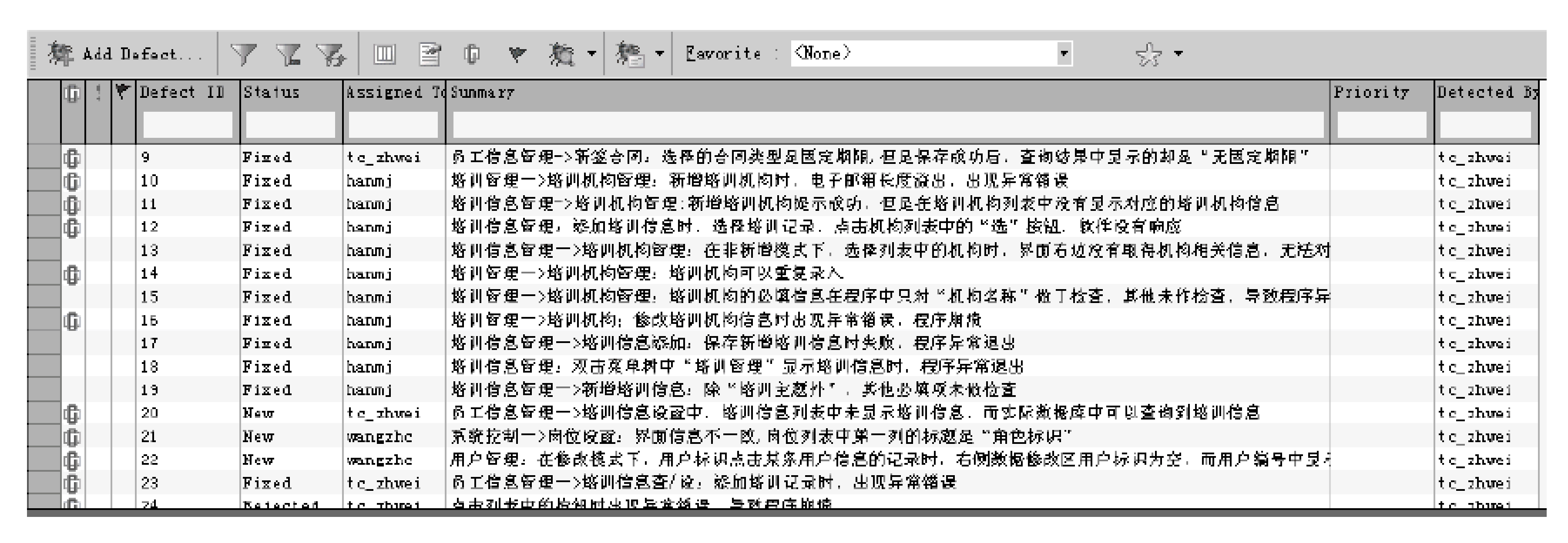


图 18-11　缺陷管理

18.2　Bugzilla

18.2.1　工具介绍

Bugzilla 是一款共享的免费的产品缺陷记录及跟踪工具。由 Mozilla 公司提供。创

始人是 Terry Weissman，开始时使用一种名为“TCL”的语言创建的，后用 Perl 语言实现，并作为 Open source 发布。Bugzilla 能够为你建立一个完善的 bug 跟踪体系，包括报告 bug、查询 bug 记录并产生报表、处理解决 bug、管理员系统初始化和设置四部分。

18.2.2 Bugzilla 功能特点

Bugzilla 具有如下特点。

（1）基于 Web 方式，安装简单、运行方便快捷、管理安全。

（2）有利于缺陷的清楚传达。本系统使用数据库进行管理，提供全面详尽的报告输入项，产生标准化的 bug 报告。提供大量的分析选项和强大的查询匹配能力，能根据各种条件组合进行 bug 统计。当缺陷在它的生命周期中变化时，开发人员、测试人员、管理人员将及时获得动态的变化信息，允许获取历史记录，并在检查缺陷的状态时参考这一记录。

（3）系统灵活，强大的可配置能力。Bugzilla 工具可以对软件产品设定不同的模块，并针对不同的模块设定开发人员和测试人员。这样可以实现提交报告时自动发给指定的责任人，并可设定不同的小组，权限也可划分。设定不同的用户对 bug 记录的操作权限不同，可有效控制测试人员和开发人员对缺陷状态的变更管理。允许设定不同的严重程度和优先级。可以在缺陷的生命期中管理缺陷。从最初的报告到最后的解决，确保了缺陷不会被忽略。同时可以使注意力集中在优先级和严重程度高的缺陷上。

（4）自动发送 E-mail，通知相关人员。根据设定的不同责任人，自动发送最新的动态信息，有效地帮助测试人员和开发人员进行沟通。

本章小结

本章主要介绍了一种典型的测试管理工具 TestDirector 的使用方法，并结合项目给出了应用实例。

TestDirector 是 HP 公司一个测试管理工具，是业界第一个基于 Web 的测试管理系统，它可以在您公司内部或外部进行全球范围内测试的管理，包括需求管理，测试计划，测试执行以及错误跟踪等功能，TestDirector 极大地加速了测试过程。

同时，本章对于 Bugzilla 这样一个免费的测试管理工具也作了简单介绍，Bugzilla 因其投资成本低而具备一定的优势，在很多组织也有一定的应用基础。

事实上，随着软件测试产业发展，对于测试管理的迫切性开始浮现，因其介入门槛相对低一些，因此也引来了大批后来者加入这方面的研究，产品也日渐丰富，我们的选择空间越来越大，测试同行完全可以根据个人的需要去为自己的组织选择一款合适的测试管理平台。



案例项目业务及技术背景介绍

企业或组织在市场中的竞争优势集中表现为以下两点：

① 企业是否具有在人才市场中具有优势的人才。

② 企业所具有的人才是否具有合适的环境。

现在的成功企业一般关注两个方面：一是积极寻找合适的人才，并想办法留住人才；二是营造企业的内部环境，一面促进人才的成长，一面又有利于人才脱颖而出。这两个方面，也许后者更重要，所以现在企业最流行的莫过于重组与再造，而且所有的管理活动都可以通过计算机和网络完成，比如生产管理系统、物流管理系统、财务管理系统、客户管理系统等。

这些管理系统都是为了加快企业应变能力和业务处理能力，但所有这些的实现又以企业的员工为基础，所以人力资源管理者，如果还想成为CEO的战略伙伴，就必须要改变自己的工作模式，加快自身对企业内外环境、企业目标，以及其他直线部门的需要和变化的响应，并提高工作质量。要想实现对这种快速变化的环境和需求的快速响应，我们就必须能够尽早获得和传输需求信息或变化的信息，必须更快地处理这种信息，并更快地做出相应的处理，以满足或处理这种变化。所以人力资源管理系统，也就成为了越来越多企业不得已的选择，也是必然的选择。

本书中采用的人力资源管理系统(英文标识HRMIS，以下相同处均以HRMIS代替)是我们从当前主流人力资源管理系统的基本功能集中采样实现，作为一个实际项目案例，主要完成了员工信息管理、工作履历、合同信息以及培训等信息处理工作。

C/S和B/S是当今软件市场的主流结构，其中具备C/S结构的软件系统可以充分利用两端硬件环境的优势，将任务合理分配到Client端和Server端来实现，降低了系统的通信开销。本书中讲述的HRMIS是基于C/S模式的应用，客户软件通过ADO共同访问MySQL数据库服务器，实现信息的处理和共享。HRMIS业务逻辑采用UI(界面)－PROCESS UNIT(逻辑处理封装单元)－DB(数据操作封装)三层封装的设计模式，以降低代码复杂度，提高代码可读性和复用度。

1. ADO

ADO(ActiveX Data Objects)是美国微软公司于1996年发布的一个用于存取数据

源的 COM 组件。它提供了编程语言和统一数据访问方式 OLE DB 的一个中间层。允许开发人员编写访问数据的代码而不用关心数据库是如何实现的，而只用关心到数据库的连接。

2．MySQL

MySQL 是现今最流行的开放源码的 SQL 数据库管理系统，它是由瑞典 MySQL AB 公司开发、发布并支持的。MySQL 具有快速、可靠和易于使用的特点，MySQL 服务器可工作在客户端/服务器模式下，或嵌入式系统中。

MySQL 一个开放源代码项目，可以在绝大多数的情况下免费使用，具备低成本特点，这使得它在开源社区中被广泛地使用。在本书的案例项目 HRMIS 中采用的是最新版 MySQL6.0。

【案例项目的文档编写标准说明】

本书中采用的 HRMIS 开发类文档提供了《系统需求说明书》（文档编号：HR-SRS-0200-200811）和《系统设计说明书》（文档编号：HRMIS-SDS-0100-200812）这两个文档，主要内容依据 GB/T 8567—2006《计算机软件文档编制规范》编写。

HRMIS 测试类文档提供了测试计划、测试需求、缺陷报告、测试报告等，这些文档的主要编写依据是 GB/T GB 9386－2008《计算机软件测试文档编制规范》，同时也参考了有关第三方软件测试实验室管理标准（如 ISO/IEC 17025）的有关文档编写要求以及测试工作中出现的一些所谓最佳实践，力争使本书所提供的资料具备较高的权威性、代表性，同时也具备普遍的适用性。

APPENDIX

附录B 软件工程国家标准目录

我国自20世纪80年代以来发布了多项软件工程标准：

序号	国家标准编号	年份	标 准 名 称
1	GB/T1526	1989	信息处理数据流程图、程序流程图、系统流程图、程序网络图和系统资源图的文件编制符号及约定
2	GB/T8566	2007	信息技术 软件生存周期过程
3	GB/T8567	2006	计算机软件文档编制规范
4	GB/T9385	2008	计算机软件需求规格说明规范
5	GB/T9386	2008	计算机软件测试文档编制规范
6	GB/T11457	2006	软件工程术语
7	GB/T 13502	1992	信息处理 程序构造及其表示的约定
8	GB/T14085	1993	信息处理系统 计算机系统配置图符号及其约定
9	GB/T14394	2008	计算机软件可靠性和维护性管理
10	GB/T15532	2008	计算机软件测试规范
11	GB/T15535	1995	信息处理 单命中判定表规范
12	GB/T16260.1	2006	软件工程 产品质量 第1部分：质量模型
13	GB/T16260.2	2006	软件工程 产品质量 第2部分：外部度量
14	GB/T16260.3	2006	软件工程 产品质量 第3部分：内部度量
15	GB/T16260.4	2006	软件工程 产品质量 第4部分：使用质量的度量
16	GB/T16680	1996	软件文档管理指南
17	GB/T17544	1998	信息技术 软件包 质量要求和测试
18	GB/T18234	2000	信息技术 CASE工具的评价与选择指南
19	GB/T18491.1	2001	信息技术 软件测量 功能规模测量 第1部分：概念定义
20	GB/T18492	2001	信息技术 系统及软件完整性级别

续表

序号	国家标准编号	年份	标 准 名 称
21	GB/Z18493	2001	信息技术 软件生存周期过程指南
22	GB/T18905.1	2002	软件工程 产品评价 第 1 部分：概述
23	GB/T18905.2	2002	软件工程 产品评价 第 2 部分：策划和管理
24	GB/T18905.3	2002	软件工程 产品评价 第 3 部分：开发者用的过程
25	GB/T18905.4	2002	软件工程 产品评价 第 4 部分：需方用的过程
26	GB/T18905.5	2002	软件工程 产品评价 第 5 部分：评价者用的过程
27	GB/T18905.6	2002	软件工程 产品评价 第 6 部分：评价模块的文档编制
28	GB/Z18914	2002	信息技术 软件工程 CASE 工具的采用指南
29	GB/Z20156	2006	软件工程 软件生存周期过程 用于项目管理的指南
30	GB/T20157	2006	软件工程 软件维护
31	GB/T20158	2006	信息技术 软件生存周期过程 配置管理
32	GB/T20917	2007	软件工程 测量过程
33	GB/T20918	2007	信息技术 软件生存周期过程 风险管理

参考文献

[1] 周之英.现代软件工程 第1册.北京：科学出版社，1999.
[2] 梅宏(译).软件工程——实践者的研究方法.北京：机械工业出版社，2006.
[3] Capability Maturity Model Integration (CMMISM)，Version 1.1，Continuous Representation，
[4] Capability Maturity Model Integration (CMMISM)，Version 1.1，Staged Representation，
[5] 柳纯录，黄子河，陈渌萍.软件评测师教程.北京：清华大学出版社，2006.
[6] Capers Jones. Softuare Assessments，Bench marks，and Best Pracfices. Addison Wesley，2000.
[7] http://ks.cn.yahoo.com/question/1406080709088.html.
[8] http://se.csai.cn/testmanage/200802011051241856.htm.
[9] http://se.csai.cn/ISO/200610250929491262.htm.
[10] http://tech.it168.com/a2008/0724/198/000000/98539.shtmt

注：本书在编写过程中部分内容参考了已公开发表的书目和研究资料，另外还有来源于网络的软件测试从业者的经验之谈，其中有些内容在转载过程中原作者已不可考，在此一并列出，向原作者、为本书提供评审和积极建议的人员以及其他所有关心本书并为本书的最终形成有贡献的人表示诚挚的感谢。

读者意见反馈

亲爱的读者：

感谢您一直以来对清华版计算机教材的支持和爱护。为了今后为您提供更优秀的教材，请您抽出宝贵的时间来填写下面的意见反馈表，以便我们更好地对本教材做进一步改进。同时如果您在使用本教材的过程中遇到了什么问题，或者有什么好的建议，也请您来信告诉我们。

地址：北京市海淀区双清路学研大厦 A 座 602　　计算机与信息分社营销室 收
邮编：100084　　电子邮件：jsjjc@tup.tsinghua.edu.cn
电话：010-62770175-4608/4409　　邮购电话：010-62786544

教材名称：软件测试技术
ISBN：978-7-302-20878-5

个人资料

姓名：________ 年龄：______ 所在院校/专业：____________

文化程度：________ 通信地址：____________________

联系电话：________ 电子信箱：____________________

您使用本书是作为：□指定教材 □选用教材 □辅导教材 □自学教材

您对本书封面设计的满意度：

□很满意 □满意 □一般 □不满意　改进建议____________

您对本书印刷质量的满意度：

□很满意 □满意 □一般 □不满意　改进建议____________

您对本书的总体满意度：

从语言质量角度看 □很满意 □满意 □一般 □不满意

从科技含量角度看 □很满意 □满意 □一般 □不满意

本书最令您满意的是：

□指导明确 □内容充实 □讲解详尽 □实例丰富

您认为本书在哪些地方应进行修改？（可附页）

____

____

您希望本书在哪些方面进行改进？（可附页）

____

____

电子教案支持

敬爱的教师：

为了配合本课程的教学需要，本教材配有配套的电子教案（素材），有需求的教师可以与我们联系，我们将向使用本教材进行教学的教师免费赠送电子教案（素材），希望有助于教学活动的开展。相关信息请拨打电话 010-62776969 或发送电子邮件至 jsjjc@tup.tsinghua.edu.cn 咨询，也可以到清华大学出版社主页（http://www.tup.com.cn 或 http://www.tup.tsinghua.edu.cn）上查询。